Revocation for Delegatable Anonymous Credentials

MSR-TR-2010-170⋆

Tolga Acar and Lan Nguyen

Microsoft Research
One Microsoft Way, Redmond, WA 98052, USA
tolga@microsoft.com, languyen@microsoft.com
http://research.microsoft.com/en-us/people/{tolga,languyen}

Abstract. This paper introduces and formalizes *homomorphic proofs* that allow 'adding' proofs and proof statements to get a new proof of the 'sum' statement. Additionally, we introduce a construction of homomorphic proofs, and show an *accumulator scheme with delegatable non-membership proofs* (ADNMP) as one of its applications with provable security. Finally, the proposed accumulator method extends the BC-CKLS scheme [1] to create a new provably secure *revocable delegatable anonymous credential* (RDAC) system. Intuitively, the new accumulator's delegatable non-membership (NM) proofs enable user A, without revealing her identity, to delegate to user B the ability to prove that A's identity is not included in a blacklist that can later be updated. The delegation is redelegatable, unlinkable, and verifiable.

1 Introduction

Proof systems play important roles in many cryptographic systems, such as signature, authentication, encryption, anonymous credential and mix-net. In a proof system between a prover and a verifier, an honest prover with a *witness* can convince a verifier about the truth of a *statement* but an adversary cannot convince a verifier of a false statement. Groth and Sahai [2] proposed a novel class of non-interactive proof systems (GS) with a number of desirable properties which are not available in previous ones. They are efficient and general. They do not require the random oracle assumption [3]. They can be randomized, i.e. one can generate a new proof from an existing proof of the same statement without knowing the witness. In this paper, we unveil another valuable feature of GS proofs: homomorphism.

Proof systems are used to construct *accumulators* [4–8]. An accumulator allows aggregation of a large set of elements into one constant-size *accumulator value*. The 'membership' proof system proves that an element is accumulated. An accumulator is *universal* if it has 'non-membership' proof system to prove that a given element is not accumulated in the accumulator value [9, 10]. An

⋆ The proceedings version is published at PKC 2011.

accumulator is *dynamic* if the costs of adding and deleting elements and updating the accumulator value and proof systems' witnesses do not depend on the number of elements aggregated. Applications of accumulators include space-efficient time stamping, ad-hoc anonymous authentication, ring signatures, ID-Based systems, and membership *revocation* for identity escrow, group signatures and *anonymous credentials* [6].

In *anonymous credential* systems, a user can prove some credentials without revealing any other private information such as her identity. There have been several proposals [11, 12, 1]; applications such as in direct anonymous attestation (DAA) [13] and anonymous electronic identity (eID) token [14, 15]; and implementations such as U-prove [15], Idemix [14] and java cards [16]. An anonymous credential system is *delegatable* [1] if its credential can be delegated from one user to another user so that a user can anonymously prove a credential which is delegated some levels away from the original issuer. Delegation is important for efficient credential management in organizations, as a person typically delegates certain authorities to colleagues to execute tasks on her behalf. *Revocation* is indispensable in credential systems in practice, as dispute, compromise, abuse, mistake, identity change, hacking and insecurity can make any credential become invalid before its expiration. The anonymity and delegation properties make revocation more challenging: the user must prove anonymously that her whole credential chain is not revoked. The primary revocation methods are based on accumulators [17, 10], offering a constant cost for an unrevoked proof. However, the current schemes do not work for delegated anonymous credentials.

Contributions. We present three contributions in this paper, incrementally building on each other: *(i)* formal definition of homomorphic proofs and a construction based on GS proofs, *(ii)* dynamic universal accumulators with delegatable non-membership proof (ADNMP), and *(iii)* a revocable delegatable anonymous credential system (RDAC).

We first introduce and formally define the new notion of *homomorphic proofs*, which means there is an operation that 'adds' proofs, their statements and witnesses to produce a new proof of the 'sum' statement and the 'sum' witness. We present and prove a construction for homomorphic proofs from GS proofs [2]. The general nature of GS proofs partly explains the reason behind its numerous applications: group signatures, ring signatures, mix-nets, anonymous credentials, and oblivious transfers. Our homomorphic construction uses the most general form of GS proofs to maximize the range of possible applications.

Homomorphic proofs can be applied to homomorphic signatures [18], homomorphic authentication [19], that found applications in provable cloud storage [19], network coding [20, 21], digital photography [22] and undeniable signatures [23]. Another possible application area is homomorphic encryption and commitment schemes that are used in mix-nets [24], voting [25], anonymous credentials [1] and other multi-party computation systems. Gentry's recent results on fully homomorphic encryption [26] allow computing any generic function of encrypted data without decryption and can be applied to cloud computing and searchable encryption.

Section 3.3 compares this work to the DHLW homomorphic NIZK (Non Interactive Zero Knowledge) recently proposed in [27]. While the DHLW scheme takes the traditional homomorphism approach, we employ Abelian groups and introduce a more general definition where proof systems satisfying the DHLW definition are a subset of the new proof systems. We note that DHLW's homomorphic NIZK definition and construction do not cover the new homomorphic proofs to build ADNMP and RDAC. From an application point of view, DHLW homomorphic NIZK targets leakage-resilient cryptography, and the new homomorphic proofs target accumulators and revocation.

Secondly, we introduce and build an *accumulator with delegatable non-membership proof* (ADNMP) scheme based on homomorphic proofs. We define security requirements for ADNMP, and give security proofs for the ADNMP scheme. The constructions in the SXDH (Symmetric External Diffie Hellman) or SDLIN (Symmetric Decisional Linear) instantiations of GS proofs allow the use of the most efficient curves for pairings in the new accumulator scheme [28].

To our knowledge, this is the first accumulator with a *delegatable* non-membership proof. Previously, there were only two accumulators with non-membership proofs, i.e. universal accumulators LLX [9] and ATSM [10]; both are not delegatable. Delegability allows us to construct delegatable revocation for delegatable anonymous credentials. Our accumulator uses GS proofs without random oracles where LLX and ATSM rely on the random oracle assumption for non-interactive proofs. LLX is based on the Strong RSA assumption and defined in composite-order groups, and ATSM is based on the Strong DH assumption and defined in prime-order bilinear pairing groups. Our scheme is also built in prime-order bilinear pairing groups that require storage much smaller than RSA composite-order groups. The new non-membership prover requires no pairing compared to ATSM's four pairings.

The main challenge in blacklisting delegatable anonymous credentials that can further be delegated is to create accumulators satisfying the following requirements. First, user A, without revealing private information, can delegate the ability to prove that her identity is not accumulated in any blacklist to user B so that such proofs generated by A and B are indistinguishable and the blacklist may change anytime. Second, the delegation must be unlinkable, i.e. it must be hard to tell if two such delegations come from the same delegator A. Third, user B is able to redelegate the ability to prove that A's credential is not blacklisted to user C, such that the information C obtains from the redelegation is indistinguishable from the information one obtains from A's delegation. Finally, any delegation information must be verifiable for correctness. The new ADNMP scheme satisfies these requirements.

By employing the ADNMP approach, our final contribution is to create the first *delegatable anonymous credential system with delegatable revocation* capability; an RDAC system. Traditionally, blacklisting of anonymous credentials relies on accumulators [8]. The identities of revoked credentials are accumulated in a blacklist, and verification of the accumulator's NM proof determines the credential's revocation status. A natural rule in a revoked delegatable credential, that our scheme also follows, is to consider all delegated descendants of the credential revoked. Applying that rule to delegatable anonymous credentials, a user must anonymously prove that all ancestor credentials are not revoked, even when the blacklist changes.

Homomorphic proofs bring delegability of proofs to another level. A proof's *statement* often consists of *commitments* of variables (witnesses) and *conditions*. Randomizable and malleable proofs introduced in [1] allows generation of a new proof and randomization of the statement's commitments without knowing the witness, but the statement's conditions always stay the same. Homomorphic proofs allow generating a new proof for a new statement containing new conditions, without any witness. A user can delegate her proving capability to another user by revealing some homomorphic proofs. A linear combination of these proofs and their statements allows the delegatee to generate new proofs for other statements with different conditions (e.g., an updated blacklist in ADNMP). In short, the BCCKLS paper [1] deals with delegating proofs of the same statements' conditions, whereas this paper deals with delegating proofs of changing statements' conditions.

2 Background

Appendix 8 provides more details of existing cryptographic primitives: Bilinear Map Modules, $\mathcal{R}$-module, Bilinear pairings, SXDH, Composable zero-knowledge (ZK), Randomizing proofs and commitments, Partial extractability, Accumulator, and Delegatable anonymous credentials.

NOTATION. PPT stands for Probabilistic Polynomial Time; CRS for Common Reference String; Pr for Probability; NM for non-membership; ADNMP for Accumulator with Delegatable NM Proofs; RDAC for Revocable Delegatable Anonymous Credential; and $\leftarrow$ for random output. For a group $\mathbb{G}$ with identity $\mathcal{O}$, let $\mathbb{G}^* := \mathbb{G}\backslash\{\mathcal{O}\}$. $Mat_{m\times n}(\mathcal{R})$ is the set of matrices with size $m \times n$ in $\mathcal{R}$. For a matrix Γ, $\Gamma[i,j]$ is the value in i^{th} row and j^{th} column. A vector $\vec{z}$ of l elements can be viewed as a matrix of l rows and 1 column. For a vector $\vec{z}$, $z[i]$ is the i^{th} element. For a function $\nu : \mathbb{Z} \to \mathbb{R}$, ν is *negligible* if $|\nu(k)| < k^{-\alpha}$, $\forall\alpha > 0$, $\forall k > k_0$, $\exists k_0 \in \mathbb{Z}^+$, $k \in \mathbb{Z}$.

PROOF SYSTEM. Let $\mathbf{R}$ be an efficiently computable relation of ($Para$, Sta, Wit) with setup parameters $Para$, a statement Sta, and a witness Wit. A non-interactive proof system for $\mathbf{R}$ consists of 3 PPT algorithms: a Setup, a prover Prove, and a verifier Verify. A non-interactive proof system (Setup, Prove, Verify) must be complete and sound. $\mathsf{Completeness}$ means that for every PPT adversary $\mathcal{A}$, $|\Pr[Para \leftarrow \mathsf{Setup}(1^k); (Sta, Wit) \leftarrow \mathcal{A}(Para); Proof \leftarrow \mathsf{Prove}(Para, Sta, Wit) : \mathsf{Verify}(Para, Sta, Proof) = 1 \text{ if } (Para, Sta, Wit) \in \mathbf{R}] - 1|$ is negligible. $\mathsf{Soundness}$ means that for every PPT adversary $\mathcal{A}$, $|\Pr[Para \leftarrow \mathsf{Setup}(1^k); (Sta, Proof) \leftarrow \mathcal{A}(Para) : \mathsf{Verify}(Para, Sta, Proof) = 0 \text{ if } (Para, Sta, Wit) \notin \mathbf{R}, \forall Wit] - 1|$ is negligible.

GS PROOFS. Appendix 8.2 provides a comprehensive summary of GS proofs and its instantiation in SXDH. Briefly, the GS *setup* algorithm generates Gk

and CRS σ. Gk contains L tuples, each of which has the form (A_1, A_2, A_T, f) where A_1, A_2, A_T are $\mathcal{R}$-modules with map $f : A_1 \times A_2 \to A_T$. L is also the number of equations in a statement to be proved. CRS σ contains L corresponding tuples of $\mathcal{R}$-modules and maps $(B_1, B_2, B_T, \iota_1, \iota_2, \iota_T)$, where $\iota_j : A_j \to B_j$. A GS *statement* is a set of L corresponding tuples ($\vec{a} \in A_1^n, \vec{b} \in A_2^m, \Gamma \in Mat_{m\times n}(\mathcal{R}), t \in A_T$) satisfying $\vec{a} \cdot \vec{y} + \vec{x} \cdot \vec{b} + \vec{x} \cdot \Gamma\vec{y} = t$; where ($\vec{x} \in A_1^m, \vec{y} \in A_2^n$) is the corresponding witness (there are L witness tuples), and denote $\vec{a} \cdot \vec{y} = \sum_{j=1}^{n} f(a[j], y[j])$. The *proof* of the statement includes L corresponding tuples, each of which consists of commitments $\vec{c} \in B_1^m$ of $\vec{x}$ and $\vec{d} \in B_2^n$ of $\vec{y}$ with values $\vec{\pi}$ and $\vec{\psi}$. In the SXDH instantiation of GS proofs, *Para* includes bilinear pairing setup $Gk = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, P_1, P_2)$ and CRS $\sigma = (B_1, B_2, B_T, F, \iota_1, p_1, \iota_2, p_2, \iota'_1, p'_1, \iota'_2, p'_2, \iota_T, p_T, \vec{u}, \vec{v})$ where $B_1 = \mathbb{G}_1^2$, $B_2 = \mathbb{G}_2^2$ and $B_T := \mathbb{G}_T^4$. The maps are $\iota_j : A_j \to B_j$, $p_j : B_j \to A_j$, $\iota'_j : \mathbb{Z}_p \to B_j$ and $p'_j : B_j \to \mathbb{Z}_p$. Vectors $\vec{u}$ of $u_1, u_2 \in B_1$ and $\vec{v}$ of $v_1, v_2 \in B_2$ are commitment keys for $\mathbb{G}_1$ and $\mathbb{G}_2$.

3 Homomorphic Proofs

3.1 Formalization

Recall that an Abelian group must satisfy 5 requirements: Closure, Associativity, Commutativity, Identity Element and Inverse Element.

Definition 1. *Let* $(\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ *be a proof system for a relation* $\mathbf{R}$ *and* $Para \leftarrow \mathsf{Setup}(1^k)$. *Consider a subset* Π *of all* $(Sta, Wit, Proof)$ *such that* $(Para, Sta, Wit) \in \mathbf{R}$ *and* $\mathsf{Verify}(Para, Sta, Proof) = 1$, *and an operation* $+_\Pi : \Pi \times \Pi \to \Pi$. Π *is a set of* ***homomorphic*** *proofs if* $(\Pi, +_\Pi)$ *satisfies the 3 requirements: Closure, Associativity and Commutativity.*

Consider an $I_\Pi := (Sta_0, Wit_0, Proof_0) \in \Pi$. Π *is a set of* ***strongly homomorphic*** *proofs if* $(\Pi, +_\Pi, I_\Pi)$ *forms an Abelian group where* I_Π *is the identity element.*

Note that if Π is strongly homomorphic, then Π is also homomorphic. If $+_\Pi$ $((Sta_1, Wit_1, Proof_1), (Sta_2, Wit_2, Proof_2)) \mapsto (Sta, Wit, Proof)$, we have the following notations:

$(Sta, Wit, Proof) \leftarrow (Sta_1, Wit_1, Proof_1) +_\Pi (Sta_2, Wit_2, Proof_2)$, $Sta \leftarrow Sta_1 +_\Pi Sta_2$, $Wit \leftarrow Wit_1 +_\Pi Wit_2$, and $Proof \leftarrow Proof_1 +_\Pi Proof_2$.

We also use the multiplicative notation $n(Sta, Wit, Proof)$ for the self addition for n times of $(Sta, Wit, Proof)$. Similarly, we also use $\sum_i n_i$ $(Sta_i, Wit_i, Proof_i)$ to represent linear combination of statements, witnesses and proofs. These homomorphic properties are particularly useful for randomizable proofs: one can randomize a proof computed from the homomorphic operation to get another proof, which is indistinguishable from a proof generated by Prove.

3.2 GS Homomorphic Proofs

Consider a GS proof system (Setup,Prove,Verify) of L equations. Each map $\iota_i : A_i \to B_i$ satisfies $\iota_i(x_1 + x_2) = \iota_i(x_1) + \iota_i(x_2)$, $\forall x_1, x_2 \in A_1$ and $i \in \{1, 2\}$.

We first define the **identity** $I_{GS} = (Sta_0, Wit_0, Proof_0)$. Sta_0 consists of L GS equations ($\vec{a}_0$, $\vec{b}_0$, Γ_0, t_0), Wit_0 consists of L corresponding GS variables $(\vec{x}_0, \vec{y}_0)$, $Proof_0$ consists of L corresponding GS proofs $(\vec{c}_0, \vec{d}_0, \vec{\pi}_0, \vec{\psi}_0)$, and there are L tuples of corresponding maps (ι_1, ι_2). They satisfy:

- ⋄ Let m be the dimension of $\vec{b}_0$, $\vec{x}_0$ and $\vec{c}_0$. $\exists M \subseteq \{1, ..., m\}$ such that $\forall i \in M$, $b_0[i] = 0$; $\forall j \in \bar{M}$, $x_0[j] = 0$ and $c_0[j] = \iota_1(0)$, where $\bar{M} := \{1, ..., m\} \backslash M$.
- ⋄ Let n be the dimension of $\vec{a}_0$, $\vec{y}_0$ and $\vec{d}_0$. $\exists N \subseteq \{1, ..., n\}$ such that $\forall i \in N$, $a_0[i] = 0$; $\forall j \in \bar{N}$, $y_0[j] = 0$ and $d_0[j] = \iota_2(0)$, where $\bar{N} := \{1, ..., n\} \backslash N$.
- ⋄ For both $(\forall i \in \bar{M}, \forall j \in \bar{N})$ and $(\forall i \in M, \forall j \in N)$: $\Gamma_0[i, j] = 0$.
- ⋄ $t_0 = 0$, $\vec{\pi}_0 = 0$, and $\vec{\psi}_0 = 0$.

We next define a **set** Π_{GS} of tuples (Sta, Wit, $Proof$) from the identity I_{GS}. Sta consists of L GS equations $(\vec{a}, \vec{b}, \Gamma, t)$ (corresponding to Sta_0's $(\vec{a}_0, \vec{b}_0, \Gamma_0, t_0)$ with m, n, M, N); Wit consists of L corresponding GS variables $(\vec{x}, \vec{y})$; $Proof$ consists of L corresponding GS proofs $(\vec{c}, \vec{d}, \vec{\pi}, \vec{\psi})$; satisfying:

- ⋄ $\forall i \in M$, $x[i] = x_0[i]$ and $c[i] = c_0[i]$. $\forall j \in \bar{M}$, $b[j] = b_0[j]$.
- ⋄ $\forall i \in N$, $y[i] = y_0[i]$ and $d[i] = d_0[i]$. $\forall j \in \bar{N}$, $a[j] = a_0[j]$.
- ⋄ If $(i \in \bar{M}) \vee (j \in \bar{N})$, then $\Gamma[i, j] = \Gamma_0[i, j]$. That means $\forall i \in \bar{M}, \forall j \in \bar{N}$: $\Gamma[i, j] = 0$.

We finally define **operation** $+_{GS} : \Pi_{GS} \times \Pi_{GS} \to \Pi_{GS}$. For $i \in \{1, 2\}$ and $(Sta_i, Wit_i, Proof_i) \in \Pi_{GS}$, Sta_i consists of L GS equations $(\vec{a}_i, \vec{b}_i, \Gamma_i, t_i)$ corresponding to Sta_0's $(\vec{a}_0, \vec{b}_0, \Gamma_0, t_0)$, Wit_i consists of L corresponding GS variables $(\vec{x}_i, \vec{y}_i)$, and $Proof_i$ consists of L corresponding GS proofs $(\vec{c}_i, \vec{d}_i, \vec{\pi}_i, \vec{\psi}_i)$. We compute ($Sta$, Wit, $Proof$) $\leftarrow$ (Sta_1, Wit_1, $Proof_1$) $+_{GS}$ (Sta_2, Wit_2, $Proof_2$) of corresponding $(\vec{a}, \vec{b}, \Gamma, t)$, $(\vec{x}, \vec{y})$ and $(\vec{c}, \vec{d}, \vec{\pi}, \vec{\psi})$ as follows.

- ⋄ $\forall i \in M$: $x[i] := x_1[i]$; $c[i] := c_1[i]$; $b[i] := b_1[i] + b_2[i]$. $\forall j \in \bar{M}$: $b[j] := b_1[j]$; $x[j] := x_1[j] + x_2[j]$; $c[j] := c_1[j] + c_2[j]$.
- ⋄ $\forall i \in N$: $y[i] := y_1[i]$; $d[i] := d_1[i]$; $a[i] := a_1[i] + a_2[i]$. $\forall j \in \bar{N}$: $a[j] := a_1[j]$; $y[j] := y_1[j] + y_2[j]$; $d[j] := d_1[j] + d_2[j]$.
- ⋄ If $(i \in \bar{M}) \vee (j \in \bar{N})$, then $\Gamma[i, j] := \Gamma_1[i, j]$. Otherwise, $\Gamma[i, j] := \Gamma_1[i, j] + \Gamma_2[i, j]$.
- ⋄ $t = t_1 + t_2$, $\vec{\pi} = \vec{\pi}_1 + \vec{\pi}_2$, and $\vec{\psi} = \vec{\psi}_1 + \vec{\psi}_2$.

Theorem 1. *In the definitions above, Π_{GS} is a set of strongly homomorphic proofs with operation $+_{GS}$ and the identity element I_{GS}.*

Proof of theorem 1 can be found in Appendix 9.

3.3 Comparison with the DHLW homomorphic NIZK

We compare our homomorphic proof approach with the independently proposed DHLW homomorphic NIZK [27]. Intuitively, DHLW defines that a NIZK proof system is homomorphic if for any $(Para, Sta_1, Wit_1), (Para, Sta_2, Wit_2) \in \mathbf{R}$: $\mathsf{Prove}(Para, Sta_1, Wit_1)_{Rand_1} + \mathsf{Prove}(Para, Sta_2, Wit_2)_{Rand_2} = \mathsf{Prove}(Para, Sta_1 + Sta_2, Wit_1 + Wit_2)_{Rand_1+Rand_2}$, where $\mathsf{Prove}(\ldots)_{Rand}$ is the output of $\mathsf{Prove}()$ with randomness $Rand$. The new definition in this paper requires homomorphism for a subset of proofs generated by Prove, and differs from DHLW's homomorphism requirement for all such proofs, covering more proof systems.

The DHLW's homomorphic NIZK construction a special case of our construction above. It is for statements of 'one-sided' GS equations $\{\vec{x}_k \cdot \vec{b}_k = t_k\}_{k=1}^{L}$ whereas our construction generalizes to statements of 'full' GS equations $\{\vec{a}_k \cdot \vec{y}_k + \vec{x}_k \cdot \vec{b}_k + \vec{x}_k \cdot \Gamma \vec{y}_k = t_k\}_{k=1}^{L}$. As shown later, the ADNMP and RDAC are based on a GS homomorphic proof system of 'full' equations $\{(y_1 + y_2)X_{j1} + y_{j3}A_1 = T_{j1} \wedge X_{j3} - y_{j3}A_2 = 0 \wedge y_{j3}X_{j2} = T_{j2}\}_{j=1}^{m}$.

4 Accumulator with Delegatable NM Proofs - ADNMP

We refer to a universal accumulator as (Setup, ProveNM, VerifyNM, CompNMWit, Accu), that consists of only algorithms for setup; generating, verifying and computing witnesses for **non-membership** proofs; and accumulating, respectively. This paper does not deal with membership proofs. Appendix 8.3 provides more details on accumulators.

The delegating ability to prove statements allows another user to prove the statements on one's behalf without revealing the witness, even if the statements' conditions change over time. For privacy reasons, adversaries should not be able to distinguish different delegations from different users. The delegatee can verify a delegation and unlinkably redelegate the proving ability further to other users. Thus, delegating an accumulator's NM proofs should meet 4 conditions formalized in Definition 2. *Delegability* means that an element *Ele*'s owner can delegate her ability to prove that *Ele* is not accumulated without trivially revealing *Ele*. Even if the set of accumulated elements change overtime, the delegatee does not need to contact the delegator again to generate the proof. The owner gives the delegatee a key *De* generated from *Ele*. The proof generated from *De* by CompNMProof is indistinguishable from a proof generated by ProveNM. *Unlinkability* means that a delegatee should not be able to distinguish whether or not two delegating keys originate from the same element. It implies that it is computationally hard to find an element from its delegating keys. *Redelegability* means that the delegatee can redelegate *De* as De' to other users, and still maintains indistinguishability of *De* and De'. *Verifiability* means that one is able to validate that a delegating key *De* is correctly built.

Definition 2. *A universal accumulator (*Setup*,* ProveNM*,* VerifyNM*,* CompNMWit*,* Accu*) is a secure ADNMP (Accumulator with Delegatable NM Proofs) if there exist PPT algorithms*

– Dele*: takes public parameters* $Para$ *and an element* Ele *and returns its delegating key* De*;*
– Rede*: takes* $Para$ *and a delegating key* De *and returns another delegating key* De'*;*
– Vali*: takes* $Para$ *and a delegating key* De *and returns* 1 *if* De *is valid or* 0 *otherwise;*
– $\mathsf{CompNMProof}$*: takes* $Para$, De, *an accumulator set* $AcSet$ *and its accumulator value* $AcVal$ *and returns an NM proof that the element* Ele *corresponding to* De *is not accumulated in* $AcSet$*;*

satisfying:

– *Delegability: For every PPT algorithm* $(\mathcal{A}_1, \mathcal{A}_2)$, $|Pr[\ (Para,\ Aux) \leftarrow \mathsf{Setup}(1^k);\ (Ele, AcSet, state) \leftarrow \mathcal{A}_1(Para);\ AcVal \leftarrow \mathsf{Accu}(\ Para,\ AcSet);\ Wit \leftarrow \mathsf{CompNMWit}(Para,\ Ele,\ AcSet,\ AcVal);\ Proof_0 \leftarrow \mathsf{ProveNM}(Para,\ AcVal,\ Wit);\ De \leftarrow \mathsf{Dele}(Para,\ Ele);\ Proof_1 \leftarrow \mathsf{CompNMProof}(Para,\ De,\ AcSet,\ AcVal);\ b \leftarrow \{0,1\};\ b' \leftarrow \mathcal{A}_2(state,\ AcVal,\ Wit,\ De,\ Proof_b):\ (Ele \notin AcSet) \wedge b = b'] - 1/2|$ *is negligible.*
– *Unlinkability: For every PPT algorithm* $\mathcal{A}$, $|Pr[\ (Para,\ Aux) \leftarrow \mathsf{Setup}(1^k);\ (Ele_0, Ele_1) \leftarrow Dom_{Para};\ De \leftarrow \mathsf{Dele}\ (Para,\ Ele_0);\ b \leftarrow \{0,1\};\ De_b \leftarrow \mathsf{Dele}(Para,\ Ele_b); b' \leftarrow \mathcal{A}(Para,\ De,\ De_b):\ b = b'] - 1/2|$ *is negligible.*
– *Redelegability: For every PPT algorithms* $(\mathcal{A}_1, \mathcal{A}_2)$, $|Pr[\ (Para,\ Aux) \leftarrow \mathsf{Setup}(1^k);\ (Ele, state) \leftarrow \mathcal{A}_1(Para);\ De \leftarrow \mathsf{Dele}(Para, Ele);\ De_0 \leftarrow \mathsf{Dele}(\ Para,\ Ele);\ De_1 \leftarrow \mathsf{Rede}\ (Para,\ De);\ b \leftarrow \{0,1\}; b' \leftarrow \mathcal{A}_2(state, De, De_b) : b = b'] - 1/2|$ *is negligible.*
– *Verifiability: For every PPT algorithm* $\mathcal{A}$, $|Pr[\ (Para,\ Aux) \leftarrow \mathsf{Setup}(1^k);\ Ele \leftarrow \mathcal{A}(Para);\ De \leftarrow \mathsf{Dele}(Para,\ Ele):\ \mathsf{Vali}(Para, De) = 1$ *if* $Ele \in Dom_{Para}] - 1|$ *and* $|Pr[(Para, Aux) \leftarrow \mathsf{Setup}(1^k); De' \leftarrow \mathcal{A}(Para) : \mathsf{Vali}(Para, De') = 0$ *if* $De' \notin \{De | De \leftarrow \mathsf{Dele}(Para, Ele');\ Ele' \in Dom_{Para}\}] - 1|$ *are negligible.*

Unlinkability combined with Redelegability generalizes the Unlinkability definition allowing an adversary $\mathcal{A}$ access an oracle $\mathcal{O}(Para, De)$ that returns another delegating key De' of the same element corresponding to De. That means $\mathcal{A}$ can get several delegating keys of Ele_0 and of Ele_b using $\mathcal{O}$. Rede can be used for such an oracle.

For any ADNMP, given an element Ele and a delegating key De, one can tell if De is generated by Ele as follows. First, she does not accumulate Ele and uses De to prove that De's element is not accumulated. Then she accumulates Ele and tries to prove again that De's element is not accumulated. If she cannot prove that anymore, she can conclude that Ele is De's element. Due to this restriction, in ADNMP's applications, Ele should be a secret that only its owner knows. This is related to the discussion in Appendix 10.4 about the general conflict between delegability and anonymity.

5 An ADNMP Scheme

We propose a dynamic universal ADNMP. Its Setup, Accu and UpdateVal are generalized from [7, 10].

- ⋄ Setup: We need GS instantiations where GS proofs of this accumulator are composable ZK. We can use either the SXDH or SDLIN (Symmetric DLIN) [28] instantiations. We use SXDH as an example. Generate parameters $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, P_1, P_2)$ and CRS σ with perfectly binding keys for the SXDH instantiation of GS proofs (Sections 2), and auxiliary information $Aux = \delta \leftarrow \mathbb{Z}_p^*$. For the proof, generate $A \leftarrow \mathbb{G}_1$ and $\tau := \iota_2'(\delta)$. For efficient accumulating without Aux, a tuple $\varsigma = (P_1, \delta P_1, \ldots, \delta^{q+1} P_1)$ is needed, where $q \in \mathbb{Z}_p^*$. The domain for elements to be accumulated is $\mathbb{D} = \mathbb{Z}_p^* \backslash \{-\delta\}$. We have $Para = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, P_1, P_2, A, \sigma, \varsigma, \tau)$.
- ⋄ Accu: On input $AcSet = \{a_1, ..., a_Q\} \subset \mathbb{D}$, compute $m = \lceil Q/q \rceil$. If $Aux = \delta$ is available, the output $AcVal$ is a set of m component accumulator values $\{V_j\}_{j=1}^m$ computed as $V_j = \prod_{i=(j-1)q+1; i<Q}^{jq} (\delta + a_i)\delta P_1$. If Aux is not available, $AcVal$ is efficiently computable from ς and $AcSet$.
- ⋄ UpdateVal: In case $a' \in \mathbb{D}$ is being accumulated; from 1 to m, find the first V_j that hasn't accumulated q elements, and update $V_j' = (\delta + a')V_j$; if such V_j isn't found, add $V_{m+1} = (\delta + a')\delta P_1$. In case a' is removed from $AcVal$, find V_j which contains a' and update $V_j' = 1/(\delta + a')V_j$.

In previous accumulators [7, 10], the accumulator value is a single value $V = \prod_{a_i \in AcSet} (\delta + a_i)\delta P_1$ and they require that q of ς is the upper bound on the number of elements to be accumulated, i.e. $m = 1$. The above generalization, where the accumulator value is a set of V instead, relaxes this requirement and allows the ADNMP scheme to work even when q is less than the number of accumulated elements. It also allows smaller q at setup.

5.1 NM Proof

We need to prove that an element $y_2 \in \mathbb{D}$ is not in any component accumulator value V_j of $AcVal$ $\{V_j\}_{j=1}^m$. Suppose V_j accumulates $\{a_1, ..., a_k\}$ where $k \leq q$, denote $Poly(\delta) := \prod_{i=1}^k (\delta + a_i)\delta$, then $V_j = Poly(\delta)P_1$. Let y_{j3} be the remainder of polynomial division $Poly(\delta) \bmod (\delta + y_2)$ in Z_p, and X_{j1} be scalar product of the quotient and P_1. Similar to [10], the idea for constructing NM proofs is that y_2 is not a member of $\{a_1, ..., a_k\}$ if and only if $y_{j3} \neq 0$. We have the following equation between δ, y_2, y_{j3} and X_{j1}: $(\delta + y_2)X_{j1} + y_{j3}P_1 = V_j$. Proving this equation by itself does not guarantee that y_{j3} is the remainder of the polynomial division above. It also needs to prove the knowledge of $(y_{j3}P_2, y_{j3}A)$ and the following Extended Strong DH (ESDH) assumption. It is a variation of the Hidden Strong DH (HSDH) assumption [30], though it is not clear which assumption is stronger. It is in the extended uber-assumption family [31] and can be proved in generic groups, similar to HSDH.

Definition. *q-**ESDH**: Let $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, P_1, P_2)$ be bilinear parameters, $A \leftarrow \mathbb{G}_1^*$ and $\delta \leftarrow \mathbb{Z}_p^*$. Given P_1, δP_1, ..., $\delta^{q+1}P_1, A, P_2, \delta P_2$, it is computationally hard to output $(\frac{y_3}{\delta+y_2}P_1, y_2, y_3P_2, y_3A)$ where $y_3 \neq 0$.*

We will show later that if one can prove the knowledge of $(y_{j3}P_2, y_{j3}A)$ satisfying $(\delta + y_2)X_{j1} + y_{j3}P_1 = V_j$ and y_2 is accumulated in V_j but $y_{j3} \neq 0$, then she can break the assumption. To prove the knowledge of $(y_{j3}P_2, y_{j3}A)$, we need equation $X_{j3} - y_{j3}A = 0$. To verify $y_{j3} \neq 0$, we need equation $T_j = y_{j3}X_{j2}$ and the verifier checks $T_j \neq 0$. We now present the NM proof and its security in theorem 2. Proof of theorem 2 can be found in Appendix 9.

- ⋄ CompNMWit takes y_2, and for each component accumulator value V_j of $AcVal$ $\{V_j\}_{j=1}^m$, computes remainder y_{j3} of $Poly(\delta)$ mod $(\delta + y_2)$ in Z_p which is efficiently computable from $\{a_1, ..., a_k\}$ and y_2. It then computes $X_{j1} = (Poly(\delta) - y_{j3})/(\delta + y_2)P_1$, which is efficiently computable from $\{a_1, ..., a_k\}$, y_2 and ς. The witness includes y_2 and $\{(X_{j1}, X_{j3} = y_{j3}A, y_{j3})\}_{j=1}^m$. UpdateNMWit is for one V_j at a time and similar to [10] with the extra task of updating $X_{j3} = y_{j3}A$.
- ⋄ ProveNM generates $X_{j2} \leftarrow \mathbb{G}_1^*$ and outputs $T_j = y_{j3}X_{j2}$ for each V_j and a GS proof for the following equations of variables $y_1 = \delta, y_2, \{(X_{j1}, X_{j3}, X_{j2}, y_{j3})\}_{j=1}^m$.
  $\bigwedge_{j=1}^m((y_1 + y_2)X_{j1} + y_{j3}P_1 = V_j \wedge X_{j3} - y_{j3}A = 0 \wedge y_{j3}X_{j2} = T_j)$.
  Note that the prover does not need to know y_1. From τ, it is efficient to generate a commitment of δ and the proof.
- ⋄ VerifyNM verifies the proof generated by ProveNM and checks that $T_j \neq 0$, $\forall j \in \{1, \ldots, m\}$. It accepts if both of them pass or rejects otherwise.

Theorem 2. *The proof system proves that an element is not accumulated. Its soundness depends on the ESDH assumption. Its composable ZK depends on the assumption underlying the GS instantiation (SXDH or SDLIN).*

5.2 NM Proofs are Strongly Homomorphic

We can see that for the same constant A, the same variables δ, y_2 and X_{j2} with the same commitments, the set of NM proofs has the form of strongly homomorphic GS proofs constructed in Section 3. For constructing delegatable NM proofs, we just need them to be homomorphic. More specifically, 'adding' 2 proofs of 2 sets of equations (with the same commitments for δ, y_2 and X_{j2})
$\bigwedge_{j=1}^m((\delta + y_2)X_{j1}^{(1)} + y_{j3}^{(1)}P_1 = V_j^{(1)} \wedge X_{j3}^{(1)} - y_{j3}^{(1)}A = 0 \wedge y_{j3}^{(1)}X_{j2} = T_j^{(1)})$ and
$\bigwedge_{j=1}^m((\delta + y_2)X_{j1}^{(2)} + y_{j3}^{(2)}P_1 = V_j^{(2)} \wedge X_{j3}^{(2)} - y_{j3}^{(2)}A = 0 \wedge y_{j3}^{(2)}X_{j2} = T_j^{(2)})$ form a proof of equations
$\bigwedge_{j=1}^m((\delta + y_2)X_{j1} + y_{j3}P_1 = V_j \wedge X_{j3} - y_{j3}A = 0 \wedge y_{j3}X_{j2} = T_j)$
where $X_{j1} = X_{j1}^{(1)} + X_{j1}^{(2)}$, $X_{j3} = X_{j3}^{(1)} + X_{j3}^{(2)}$, $y_{j3} = y_{j3}^{(1)} + y_{j3}^{(2)}$, $V_j = V_j^{(1)} + V_j^{(2)}$ and $T_j = T_j^{(1)} + T_j^{(2)}$.

5.3 Delegating NM Proof

We first explain the idea behind the accumulator's delegatable NM proof construction. We write the component accumulator value $V = \prod_{i=1}^k(\delta + a_i)\delta P_1$ as

$V = \sum_{i=0}^{k} b_i \delta^{k+1-i} P_1$ where $b_0 = 1$ and $b_i = \sum_{1 \le j_1 < j_2 < \ldots < j_i \le k} \prod_{l=1}^{i} a_{j_l}$. Thus, V can be written as a linear combination of $\delta P_1, \ldots, \delta^{k+1} P_1$ in ς.

Next, we construct homomorphic proofs for $(\delta + y_2)X_1^{(i)} + y_3^{(i)} P_1 = \delta^i P_1 \wedge X_3^{(i)} - y_3^{(i)} A = 0 \wedge y_3^{(i)} X_2 = T^{(i)}$ where $i \in \{1, ..., k+1\}$. Using the same linear combination of $\delta P_1, \ldots, \delta^{k+1} P_1$ for V, we linearly combine these proofs to get a proof for $(\delta + y_2)X_1 + y_3 P_1 = V \wedge X_3 - y_3 A = 0 \wedge y_3 X_2 = T$, where $X_1 = \sum_{i=0}^{k} b_i X_1^{(k+1-i)}$, $X_3 = \sum_{i=0}^{k} b_i X_3^{(k+1-i)}$, $y_3 = \sum_{i=0}^{k} b_i y_3^{(k+1-i)}$ and $T = \sum_{i=0}^{k} b_i T^{(k+1-i)}$. This is the same as the NM proof for each of the component accumulator value provided above.

We now provide the algorithms for delegating NM proofs and its security theorem. We also add UpdateProof to be used in place of CompNMProof when possible for efficiency.

- ◇ Dele$(Para, Ele)$. For each $i \in \{1, ..., q+1\}$, compute remainder $y_3^{(i)}$ of δ^i mod $(\delta + y_2)$ in Z_p, and $X_1^{(i)} = (\delta^i - y_3^{(i)})/(\delta + y_2)P_1$, which are efficiently computable from y_2 and ς. In fact, we have $y_3^{(i)} = (-1)^i y_2^i$ and $X_1^{(i+1)} = \sum_{j=0}^{i} (-1)^j y_2^j \delta^{i-j} P_1 = \delta^i P_1 - y_2 X_1^{(i)}$ (so the cost of computing all $X_1^{(i)}$, $i \in \{1, ..., q+1\}$ is about q scalar products). Generate $X_2 \leftarrow \mathbb{G}_1^*$, the delegation key De includes $\{T^{(i)} = y_3^{(i)} X_2\}_{i=1}^{q+1}$ and a GS proof of equations $\bigwedge_{i=1}^{q+1} ((\delta + y_2)X_1^{(i)} + y_3^{(i)} P_1 = \delta^i P_1 \wedge X_3^{(i)} - y_3^{(i)} A = 0 \wedge y_3^{(i)} X_2 = T^{(i)})$.
- ◇ Rede$(Para, De)$. For each $i \in \{1, ..., q+1\}$, extract proof $Proof_i$ of $y_3^{(i)} X_2 = T^{(i)}$ in De. In each $Proof_i$, for the same $y_3^{(i)}$ and its commitment, $Proof_i$ is of homomorphic form. So generate $r \leftarrow Z_p^*$ and compute $Proof_i' = rProof_i$ which is a proof of $y_3^{(i)} X_2' = T'^{(i)}$, where $X_2' = rX_2$ and $T'^{(i)} = rT^{(i)}$. Note that commitments of $y_3^{(i)}$ stay the same. For every $i \in \{1, ..., q+1\}$, replace $T^{(i)}$ by $T'^{(i)}$ and $Proof_i$ by $Proof_i'$ in De to get a new GS proof, which is then randomized to get the output De'.
- ◇ Vali$(Para, De)$. A simple option is to verify the GS proof De. An alternative way is to use batch verification: Divide De into proofs $NMProof_i$ of $(\delta + y_2)X_1^{(i)} + y_3^{(i)} P_1 = \delta^i P_1 \wedge X_3^{(i)} - y_3^{(i)} A = 0 \wedge y_3^{(i)} X_2 = T^{(i)}$ for $i \in \{1, ..., q+1\}$. Generate $q+1$ random numbers to linearly combine $NMProof_i$s and their statements and verify the combined proof and statement.
- ◇ CompNMProof$(Para, De, AcSet, AcVal)$. Divide De into proofs $NMProof_i$ as in Vali. For each component accumulator value V of $\{a_1, ..., a_k\}$, compute b_i for $i \in \{0, ..., k\}$ as above. $NMProof_i$s belong to a set of homomorphic proofs, so compute $NMProof = \sum_{i=0}^{k} b_i NMProof_{k+1-i}$, which is a proof of $(\delta + y_2)X_1 + y_3 P_1 = V \wedge X_3 - y_3 A = 0 \wedge y_3 X_2 = T$ where X_1, X_3, y_3, T and V are as explained above.

  Extract proof $SubProof$ of $y_3 X_2 = T$ in $NMProof$. For the same y_3 and its commitment, $SubProof$ is of homomorphic form. So generate $r \leftarrow Z_p^*$ and compute $SubProof' = rSubProof$ which is a proof of $y_3 X_2' = T'$, where $X_2' = rX_2$ and $T' = rT$. Note that y_3's commitment stays the same. Replace T by T' and $SubProof$ by $SubProof'$ in $NMProof$ to get a new proof

$NMProof'$.
Concatenate those $NMProof'$ of all V in $AcVal$ and output a randomization of the concatenation.

◇ UpdateProof($Para$,De,$AcSet$,$AcVal$,$Proof$,$Opens$). $Proof$ is the proof to be updated and $Opens$ contains openings for randomizing commitments of $y_1 = \delta$ and y_2 from De to $Proof$. Suppose there is a change in accumulated elements of a component value V, we just compute $NMProof'$ for the updated V as in CompNMProof. Randomize $NMProof'$ so that its commitments of y_1 and y_2 are the same as those in $Proof$ and put it in $Proof$ in place of its old part. Output a randomization of the result.

To prove that this construction provides an ADNMP, we need the following Decisional Strong Diffie Hellman (DSDH) assumption, which is not in the uber-assumption family [31], but can be proved in generic groups similarly to the PowerDDH assumption [32]. Proof of theorem 3 is in Appendix 9.

Definition. *q-**DSDH**: Let $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, P_1, P_2)$ be bilinear parameters, $B_0, B_1 \leftarrow \mathbb{G}_1^*$, $x_0, x_1 \leftarrow \mathbb{Z}_p^*$ and $b \leftarrow \{0,1\}$. Given $B_0, x_0B_0, \ldots, x_0^qB_0$, B_1, x_bB_1, $\ldots$, $x_b^qB_1$, no PPT algorithm can output $b' = b$ with a probability non-negligibly better than a random guess.*

Theorem 3. *The accumulator is a secure ADNMP, based on ESDH, DSDH and the assumption underlying the GS instantiation (SXDH or SDLIN).*

6 Revocable Delegatable Anonymous Credentials - RDAC

6.1 Model

This is a model of RDAC systems, extended from BCCKLS [1] which is briefly described in 10. Participants include users and a Blacklist Authority (BA) owning a blacklist BL. For each credential proof, a user picks a new nym indistinguishable from her other nyms. We need another type of nym for revocation, called *r-nym*, to distinguish between two types of nyms. When an r-nym is revoked, its owner cannot prove credentials anymore. The PPT algorithms are:

- Setup(1^k) outputs public parameters $Para_{DC}$, BA's secret key Sk_{BA}, and an initially empty blacklist BL. Denote BL_e an empty blacklist.
- KeyGen($Para_{DC}$) outputs a secret key Sk and a secret r-nym Rn for a user.
- NymGen($Para_{DC}, Sk, Rn$) outputs a new nym Nym with an auxiliary key $Aux(Nym)$. A user O can become a root credential issuer by publishing a nym Nym_O and a proof that her r-nym Rn_O is not revoked that O has to update when BL changes.
- Issue($Para_{DC}, Nym_O, Sk_I, Rn_I, Nym_I, Aux(Nym_I), Cred, DeInf$, Nym_U, BL, L) $\leftrightarrow$ Obtain($Para_{DC}, Nym_O, Sk_U$, Rn_U, Nym_U, $Aux(Nym_U), Nym_I$, BL, L) lets user I issue a level $L+1$ credential to user U. Sk_I, Rn_I, Nym_I and $Cred$ are the secret key, r-nym, nym and level L credential rooted at

Nym_O of issuer I. Sk_U, Rn_U and Nym_U are the secret key, r-nym and nym of user U. I gets no output and U gets a credential $Cred_U$.
Delegation information $DeInf$ is optional. When it is included, U also gets delegation information $DeInf_U$ to later prove that r-nyms of all delegators in her chain are not revoked. If $L = 0$ then $Cred$ is omitted and $DeInf = 1$ is optionally included.

– Revoke($Para_{DC}, Sk_{BA}, Rn, BL$) updates BL so that a revoked user Rn can no longer prove, delegate or receive credentials. Denote $Rn \in BL$ or $Rn \notin BL$ that Rn is blacklisted or not, respectively.
– CredProve($Para_{DC}$, Nym_O, $Cred$, $DeInf$, Sk, Rn, Nym, $Aux(Nym)$, BL, L) takes a level L credential $Cred$, Sk, Rn and optionally $DeInf$ to output $CredProof$, which proves that: (i) a credential level L is issued to Nym's owner. (ii) Nym's Rn is not revoked. (iii)(optional, when $DeInf$ is included) all r-nyms on the credential's chain are not revoked.
– CredVerify($Para_{DC}, Nym_O, CredProof, Nym, BL, L$) verifies if $CredProof$ is a valid proof of the above statements.

The differences with the model for delegatable anonymous credentials without revocation [1] are the introductions of BA with Sk_{BA} and BL; r-nyms; delegation information $DeInf$; Revoke; and the two $CredProof$'s conditions (ii) and (iii). Note that $DeInf$'s inclusion in the algorithms is optional and allows a user the choice to either just prove that she is not blacklisted or fully prove and delegate that all users on her credential chain are not blacklisted. We can use one of traditional methods for BA to obtain r-nyms to revoke (Appendix 11).

Appendix 10 formally defines RDAC security. Briefly, there are 3 requirements extended from the security definition of delegatable anonymous credentials [1]: Correctness, Anonymity and Unforgeability. Appendix 10.4 discusses the trade offs between delegability and anonymity.

7 An RDAC scheme

7.1 Overview

We first describe intuitions of the BCCKLS delegatable anonymous credential scheme in [1], and then show how ADNMP extends it to provide revocation.

BCCKLS uses an F-Unforgeable certification secure authentication scheme $\mathcal{AU}$ of PPT algorithms AtSetup, AuthKg, Authen, VerifyAuth. AtSetup(1^k) returns public parameters $Para_{At}$, AuthKg($Para_{At}$) generates a key Sk, Authen($Para_{At}$, Sk, $\vec{m}$) produces an authenticator $Auth$ authenticating a vector of messages $\vec{m}$, and VerifyAuth($Para_{At}$, Sk, $\vec{m}$, $Auth$) accepts if and only if $Auth$ validly authenticates $\vec{m}$ under Sk. The scheme's *security* requirements include F-*Unforgeability* [12] for a bijective function F, which means ($F(\vec{m})$, $Auth$) is unforgeable without obtaining an authenticator on $\vec{m}$; and *certification security*, which means no PPT adversary, even after obtaining an authenticator by the challenge secret key, can forge another authenticator. An adversary can also have access to two oracles. $\mathcal{O}_{Authen}(Para_{At}, Sk, \vec{m})$ returns Authen($Para_{At}$, Sk, $\vec{m}$) and

$\mathcal{O}_{Certify}(Para_{At}, Sk^*, (Sk, m_2, \ldots, m_n))$ returns $\mathsf{Authen}(Para_{At}, Sk^*, (Sk, m_2, \ldots, m_n))$. BCCKLS also uses a secure two party computation protocol ($\mathsf{AuthPro}$) to obtain a NIZKPK of an authenticator on $\vec{m}$ without revealing anything about $\vec{m}$.

In BCCKLS, a user U can generate a secret key $Sk \leftarrow \mathsf{AuthKg}(Para_{At})$, and many nyms $Nym = \mathsf{Com}(Sk, Open)$ by choosing different values $Open$. Suppose U has a level $L+1$ credential from O, let $(Sk_0 = Sk_O, Sk_1, \ldots, Sk_L, Sk_{L+1} = Sk)$ be the keys such that Sk_i's owner delegated the credential to Sk_{i+1}, and let $H : \{0,1\}^* \rightarrow Z_p$ be a collision resistant hash function. $r_i = H(Nym_O, atributes, i)$ is computed for a set of attributes for that level's credential. U generates a proof of her delegated credential as
$CredProof \leftarrow \mathsf{NIZKPK}[Sk_O \text{ in } Nym_O, Sk \text{ in } Nym]$
$\{(F(Sk_O), F(Sk_1), ..., F(Sk_L), F(Sk), auth_1, ..., auth_{L+1}) :$
$\mathsf{VerifyAuth}(Sk_O, (Sk_1, r_1), auth_1) \wedge$
$\mathsf{VerifyAuth}(Sk_1, (Sk_2, r_2), auth_2) \wedge ... \wedge$
$\mathsf{VerifyAuth}(Sk_{L-1}, (Sk_L, r_L), auth_L) \wedge$
$\mathsf{VerifyAuth}(Sk_L, (Sk, r_{L+1}), auth_{L+1})\}$.

Now we show how ADNMP extends BCCKLS to provide revocation. Using ADNMP, BA's blacklist BL includes an accumulated set of revoked Rns and its accumulator value. Beside a secret key Sk, user U has a secret r-nym Rn in the accumulator's domain, and generates nyms $Nym = (\mathsf{Com}(Sk, Open_{Sk}), \mathsf{Com}(Rn, Open_{Rn}))$. ADNMP allows delegation and redelegation of a proof that an Rn is not accumulated in a blacklist $Rn \notin BL$. U generates a proof of her delegated credential and validity of the credential's chain as follows.

$$\begin{aligned}
CredProof \leftarrow \mathsf{NIZKPK}[&Sk_O \text{ in } Nym_O[1], Sk \text{ in } Nym[1], Rn \text{ in } Nym[2]]\\
&\{(F(Sk_O), F(Sk_1), F(Rn_1), ..., F(Sk_L), F(Rn_L), F(Sk), F(Rn),\\
&\quad auth_1, ..., auth_L, auth_{L+1}) :\\
&\mathsf{VerifyAuth}(Sk_O, (Sk_1, Rn_1, r_1), auth_1) \wedge (Rn_1 \notin BL) \wedge\\
&\mathsf{VerifyAuth}(Sk_1, (Sk_2, Rn_2, r_2), auth_2) \wedge (Rn_2 \notin BL) \wedge ... \wedge\\
&\mathsf{VerifyAuth}(Sk_{L-1}, (Sk_L, Rn_L, r_L), auth_L) \wedge (Rn_L \notin BL) \wedge\\
&\mathsf{VerifyAuth}(Sk_L, (Sk, Rn, r_{L+1}), auth_{L+1}) \wedge (Rn \notin BL)\}.
\end{aligned}$$

Delegability allows a user, on behalf of the user's delegators without any witness, to prove that the user's ancestor delegators are not included in a changing blacklist. The proofs a user and a delegator generates are indistinguishable from each other. Redelegability allows a user to redelegate those proofs on the delegators to the user's delegatees. Unlinkability prevents colluding users to link delegations of the same delegator. Verifiability allows a user to validate the correctness of a delegation token.

7.2 Description

The RDAC scheme has several building blocks. (i) An ADNMP with a malleable NM proof system (NMPS) of $\mathsf{AcSetup}$, $\mathsf{ProveNM}$, $\mathsf{VerifyNM}$, $\mathsf{CompNMWit}$, Accu, Dele, Rede, Vali, CompNMProof, with commitment ComNM. (ii) Those from BCCKLS, including $\mathcal{AU}$; AuthPro; H; and a malleable NIPK credential proof system (CredPS) of PKSetup, PKProve, PKVerify, RandProof, with commitment Com. (iii) A malleable proof system (EQPS), with PKSetup and AcSetup in setup, to prove that two commitments Com and ComNM commit to the same value.

Assume that a delegating key De contains a commitment of element Ele. CompNMProof and Rede randomize the commitment in De and generate Ele's commitment. Elements of the accumulator domain and the authenticator's key space can be committed by Com. The following algorithm inputs are the same as in the model and omitted.

- Setup: Use PKSetup(1^k), AtSetup(1^k) and AcSetup(1^k) to generate $Para_{PK}$, $Para_{At}$, and ($Para_{Ac}$, Aux_{Ac}). The blacklist includes an accumulated set of revoked r-nyms and its accumulator value. Output an initial blacklist BL with an empty accumulator set and its initial accumulator value, $Para_{DC} = (Para_{PK}, Para_{At}, Para_{Ac}, H)$, and $Sk_{BA} = Aux_{Ac}$.
- KeyGen: Run AuthKg($Para_{At}$) to output a secret key Sk. Output a random r-nym Rn from the accumulator's domain.
- NymGen: Generate random $Open_{Sk}$ and $Open_{Rn}$, and output nym $Nym = (\mathsf{Com}(Sk, Open_{Sk}), \mathsf{Com}(Rn, Open_{Rn}))$ and $Aux(Nym) = (Open_{Sk}, Open_{Rn})$.
- The credential originator O publishes a Nym_O and a proof $NMProof_O$ that Rn_O is not revoked. O updates the proof when BL changes.
- Issue $\leftrightarrow$ Obtain: If $L = 0$ and $Nym_O \neq Nym_I$, aborts. Issue aborts if $Nym_I \neq (\mathsf{Com}(Sk_I, Open_{Sk_I}), \mathsf{Com}(Rn_I, Open_{Rn_I}))$ or PKVerify($Para_{PK}$, $(Nym_0, (\mathsf{Com}(Sk_I, 0), \mathsf{Com}(Rn_I, 0))), Cred$) rejects, or $Rn_I \in BL$, or Nym_U is invalid. Obtain aborts if $Nym_U \neq (\mathsf{Com}(Sk_U, Open_{Sk_U})$, $\mathsf{Com}(Rn_U, Open_{Rn_U}))$ or $Rn_U \in BL$. Otherwise, each of Issue and Obtain generates a proof and verifies each other's proofs that $Rn_I \notin BL$ and $Rn_U \notin BL$ using (ProveNM, VerifyNM) with EQPS (to prove that Com(Rn_I) in Nym_I and ComNM(Rn_I) generated by ProveNM commit to the same value Rn_I, and similarly for Rn_U). They then both compute $r_{L+1} = H(Nym_O, attributes, L+1)$ for a set of attributes for that level's credential. They run AuthPro for the user U to receive: $Proof_U \leftarrow$ NIZKPK[Sk_I in $Nym_I[1]$, Sk_U in $\mathsf{Com}(Sk_U, 0)$, Rn_U in $\mathsf{Com}(Rn_U, 0)$] $\{(F(Sk_I), F(Sk_U), F(Rn_U), auth)$: VerifyAuth(Sk_I, $(Sk_U, Rn_U, r_{L+1}), auth)\}$. U's output is $Cred_U = Proof_U$ when L = 0. Otherwise, suppose the users on the issuer I's chain from the root are 0 (same as O), 1, 2,..., L (same as I). I randomizes $Cred$ to get a proof $CredProof_I$ (containing the same Nym_I) that for every Nym_j on I's chain ($j \in \{1, ..., L\}$), Sk_j and Rn_j are authenticated by Sk_{j-1} (with r_j). U verifies that PKVerify($Para_{PK}, (Nym_0, Nym_I), CredProof_I$) accepts, then concatenates $Proof_U$ and $CredProof_I$ and projects Nym_I from statement to proof to get $Cred_U$.
  The optional $DeInf$ includes a list of delegating keys De_js generated by the accumulator's Dele to prove that each Rn_j is not accumulated in the blacklist, and a list of $EQProof_j$ for proving that two commitments of Rn_j in $Cred$ and De_j commit to the same value Rn_j, for $j \in \{1, ..., L-1\}$. Verifying $DeInf$ involves checking Vali($Para_{Ac}, De_j$) and $EQProof_j$, for $j \in$

$\{1,...,L-1\}$. When $DeInf$ is in the input, Issue would aborts without interacting with Obtain if verifying $DeInf$ fails. Otherwise, it uses CompNMProof to generate a proof $NMChainProof$ that each Rn_j's on I's chain of delegators is not accumulated in the blacklist. U aborts if its verification on $NMChainProof$ fails. Otherwise, I Redes these delegating keys, randomizes $EQProof_j$ to match commitments in the new delegating keys and $Cred_U$, and adds a new delegating key De_I to prove that Rn_I is not revoked and a proof $EQProof_I$ that two commitments of Rn_I in $Nym_I[2]$ and De_I commit to the same value. The result $DeInf_U$ is sent to and verified by U.

- Revoke: Add Rn to the accumulated set and update the accumulator value.
- CredProve: Abort if $Nym \neq (\mathsf{Com}(Sk, Open_{Sk}), \mathsf{Com}(Rn, Open_{Rn}))$, or $\mathsf{PKVerify}(Para_{PK}, (Nym_0, (\mathsf{Com}(Sk,0), \mathsf{Com}(Rn,0))), Cred)$ rejects, or verifying $DeInf$ fails. Otherwise, use ProveNM to generate a proof $NMProof$ that Rn is not blacklisted. Generate $EQProof'_L$ that Rn's commitments in $NMProof$ and in $Nym[2]$ both commit to the same value. Randomize $Cred$ to get a proof which contains Nym. Concatenate this proof with $NMProof$ and $EQProof'_L$ to get $CredProof'$. If the optional $DeInf$ is omitted, just output $CredProof'$.
  Otherwise, use CompNMProof to generate a proof $NMChainProof$ that each Rn_j's on the user's chain of delegators is not accumulated in the blacklist. For $j \in \{1,...,L-1\}$, update and randomize $EQProof_j$ of $DeInf$ to get $EQProof'_j$ which proves Rn_j's commitments in $NMChainProof$ and $CredProof'$ both commit to the same value. Concatenate $NMChainProof$, $CredProof'$ and $EQProof'_j$ for $j \in \{1,...,L-1\}$ to output $CredProof$ as described in (1).
- CredVerify runs PKVerify on the randomization of $Cred$, VerifyNM on $NMProof$ and
  $NMChainProof$, and verifies $EQProof'_j$ for $j \in \{1,...,L\}$ to output accept or reject.

Theorem 4. *If the authentication scheme is F-unforgeable and certification-secure; the ADNMP is secure; CredPS, NMPS and EQPS are randomizable and composable ZK; CredPS is also partially extractable; and H is collision resistant, then this construction is a secure revocable delegatable anonymous credential system.*

Proof of theorem 4 is given in Appendix 12. Instantiation of the building blocks are given in Appendix 11. Briefly, a secure ADNMP is presented in Section 5; the BCCKLS building blocks can be instantiated as in [1]; and an EQPS can be constructed from [12, 1].

8 Background

Bilinear Pairings. Let $\mathbb{G}_1$ and $\mathbb{G}_2$ be cyclic additive groups of order prime p generated by P_1 and P_2, respectively, and $\mathbb{G}_T$ be a cyclic multiplicative group of order p. An efficiently computable bilinear pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ satisfies $e(aP, bQ) = e(P, Q)^{ab}$, $\forall P \in \mathbb{G}_1, Q \in \mathbb{G}_2, a, b \in \mathbb{Z}_p$; and $e(P_1, P_2)$ generates $\mathbb{G}_T$.

SXDH [2]. For bilinear setup $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, P_1, P_2)$ with prime p, eXternal Diffie-Hellman (XDH) assumes that the Decisional Diffie-Hellman (DDH) problem is computationally hard in one of $\mathbb{G}_1$ or $\mathbb{G}_2$. Symmetric XDH (SXDH) assumes that DDH is hard in both $\mathbb{G}_1$ and $\mathbb{G}_2$.

8.1 Non-Interactive Proof System

Let $\mathbf{R}$ be an efficiently computable relation of $(\mathit{Para}, \mathit{Sta}, \mathit{Wit})$ with setup parameters *Para*, a statement *Sta*, and a witness *Wit*. A non-interactive proof system for $\mathbf{R}$ consists of 3 PPT algorithms: a Setup, a prover Prove, and a verifier Verify. A non-interactive proof system (Setup, Prove, Verify) must be complete and sound.

Completeness means that for every PPT adversary $\mathcal{A}$, $\Pr[\mathit{Para} \leftarrow \mathsf{Setup}(1^k)$; $(\mathit{Sta}, \mathit{Wit}) \leftarrow \mathcal{A}(\mathit{Para})$; $\mathit{Proof} \leftarrow \mathsf{Prove}(\mathit{Para}, \mathit{Sta}, \mathit{Wit}) : \mathsf{Verify}(\mathit{Para}, \mathit{Sta}, \mathit{Proof}) = 1 \text{ if} (\mathit{Para}, \mathit{Sta}, \mathit{Wit}) \in \mathbf{R}]$ is overwhelming.

Soundness means that for every PPT adversary $\mathcal{A}$, $\Pr[\mathit{Para} \leftarrow \mathsf{Setup}(1^k)$; $(\mathit{Sta}, \mathit{Proof}) \leftarrow \mathcal{A}(\mathit{Para}) : \mathsf{Verify}(\mathit{Para}, \mathit{Sta}, \mathit{Proof}) = 0 \text{ if} (\mathit{Para}, \mathit{Sta}, \mathit{Wit}) \notin \mathbf{R}, \forall \mathit{Wit}]$ is overwhelming.

Zero-Knowledge. A non-interactive proof system is *Zero-Knowledge (ZK)*, if the proof does not reveal any information except proving that the statement is true.

Witness Indistinguishability (WI) requires that the verifier can not determine which witness was used in the proof.

A non-interactive proof system is *composable* ZK [2] if there exists a PPT simulation algorithm outputting a trapdoor and parameters indistinguishable from Setup's output, and under the simulated parameters, ZK holds even when the adversary knows the trapdoor. Composable ZK implies the standard ZK.

Randomizing proofs and commitments. A *randomizable* non-interactive proof system [1] has another PPT algorithm RandProof, that takes as input $(\mathit{Para}, \mathit{Sta}, \mathit{Proof})$ and outputs another valid proof Proof', which is indistinguishable from a proof produced by Prove.

A PPT *commitment* algorithm Com binds and hides a value x with a random *opening* r. Informally, a commitment scheme is *randomizable* [1] if there exists a PPT algorithm ReCom such that $\mathsf{ReCom}(\mathsf{Com}(x, r), r') = \mathsf{Com}(x, r + r')$. *Sta* and *Proof* may contain commitments of variables.

A non-interactive proof system is *malleable* [1] if it is efficient to randomize the proof and its statement's commitments to get a new proof which is valid for the new statement. When possible, *concatenation* of two proofs is a proof that merges setup parameters and all commitments and proves the combination of conditions. From a proof *Proof*, a *projected* proof is obtained by moving some commitments from the statement to *Proof*.

Partial Extractability. A non-interactive proof of knowledge (NIPK) system (Setup, Prove, Verify) is *F-extractable* [12] for a bijection F if there is a PPT extractor (ExtSetup, ExtWitness) such that ExtSetup's output $Para$ is distributed identically to Setup's output, and for every PPT adversary $\mathcal{A}$,
$\Pr[(Para, td) \leftarrow \mathsf{ExtSetup}(1^k); (Sta, Proof) \leftarrow \mathcal{A}(Para); Ext \leftarrow \mathsf{ExtWitness}(td, Sta, Proof) : \mathsf{Verify}(Para, Sta, Proof) = 1 \wedge (Para, Sta, F^{-1}(Ext)) \notin \mathbf{R}]$ is negligible.

As in [12], we use the following notations NIPK or NIZKPK (ZK for zero knowledge) for a statement consisting of commitments $C_1, ..., C_k$ of witness' variables $x_1, ..., x_k$ and some *Condition*: $Proof \leftarrow \mathsf{NIPK}[x_1 \text{ in } C_1, ..., x_k \text{ in } C_k]$ $\{F(Para, Wit) : Condition(Para, Wit)\}$.

8.2 Groth-Sahai (GS) Proofs

This general description of GS proofs is based on the GS full version paper [2] which is updated and free from previous errors [28].

Bilinear Map Modules. Given a finite commutative ring $(\mathcal{R}, +, \cdot, 0, 1)$, an abelian group $(A, +, 0)$ is an $\mathcal{R}$-module if $\forall r, s \in \mathcal{R}, \forall x, y \in A$: $(r+s)x = rx+sx \wedge r(x+y) = rx+ry \wedge r(sx) = (rs)x \wedge 1x = x$. Let A_1, A_2, A_T be $\mathcal{R}$-modules with a bilinear map $f : A_1 \times A_2 \to A_T$. Let B_1, B_2, B_T be $\mathcal{R}$-modules with a bilinear map $F : B_1 \times B_2 \to B_T$ and efficiently computable maps $\iota_1 : A_1 \to B_1$, $\iota_2 : A_2 \to B_2$ and $\iota_T : A_T \to B_T$. Maps $p_1 : B_1 \to A_1$, $p_2 : B_2 \to A_2$ and $p_T : B_T \to A_T$ are hard to compute and satisfy the commutative properties: $F(\iota_1(x), \iota_2(y)) = \iota_T(f(x, y))$ and $f(p_1(x), p_2(y)) = p_T(F(x, y))$. For $\vec{x} \in A_1^n$ and $\vec{y} \in A_2^n$, denote $\vec{x} \cdot \vec{y} = \sum_{i=1}^n f(x[i], y[i])$. For $\vec{c} \in B_1^n$ and $\vec{d} \in B_2^n$, denote $\vec{c} \bullet \vec{d} = \sum_{i=1}^n F(c[i], d[i])$.

Setup. GS parameters $Para$ includes setup Gk and CRS σ. $Gk := (\mathcal{R}, \{A_1^{(i)}, A_2^{(i)}, A_T^{(i)}, f^{(i)}\}_{i=1}^L)$ where $A_1^{(i)}, A_2^{(i)}, A_T^{(i)}$ are $\mathcal{R}$-modules with map $f^{(i)} : A_1^{(i)} \times A_2^{(i)} \to A_T^{(i)}$. L is the number of equations in a statement to be proved. $\sigma := (\{B_1^{(i)}, B_2^{(i)}, B_T^{(i)}, F^{(i)}, \iota_1^{(i)}, p_1^{(i)}, \iota_2^{(i)}, p_2^{(i)}, \iota_T^{(i)}, p_T^{(i)}, \vec{u_1}^{(i)}, \vec{u_2}^{(i)}, H_1^{(i)}, ..., H_{\eta_i}^{(i)}\}_{i=1}^L)$ where $B_1^{(i)}, B_2^{(i)}, B_T^{(i)}, F^{(i)}, \iota_1^{(i)}, p_1^{(i)}, \iota_2^{(i)}, p_2^{(i)}, \iota_T^{(i)}, p_T^{(i)}$ are described above. $\vec{u_1}^{(i)}$ consists of $\hat{m}^{(i)}$ elements in $B_1^{(i)}$ and $\vec{u_2}^{(i)}$ consists of $\hat{n}^{(i)}$ elements in $B_2^{(i)}$. They are commitment keys for $A_1^{(i)}$ and $A_2^{(i)}$ respectively, as we will discuss more later. Matrices $H_1^{(i)}, ..., H_{\eta_i}^{(i)} \in Mat_{\hat{m}^{(i)} \times \hat{n}^{(i)}}(\mathcal{R})$ generate all matrices $H^{(i)}$ satisfying $\vec{u_1}^{(i)} \bullet H^{(i)} \vec{u_2}^{(i)} = 0$. It may happens that $A_k^{(i)} \equiv A_l^{(j)}$ for some $k, l \in \{1, 2\}$ and $i, j \in \{1, ..., L\}$. In that case, it is required that $(B_k^{(i)}, \iota_k^{(i)}, p_k^{(i)}, \vec{u_k}^{(i)}) \equiv (B_l^{(j)}, \iota_l^{(j)}, p_l^{(j)}, \vec{u_l}^{(j)})$.

Statement. A GS statement is a set of L equations. Each equation is over $\mathcal{R}$-modules A_1, A_2, A_T with map $f : A_1 \times A_2 \to A_T$ as follows

$$\sum_{j=1}^{n} f(a_j, y_j) + \sum_{i=1}^{m} f(x_i, b_i) + \sum_{i=1}^{m} \sum_{j=1}^{n} \gamma_{ij} f(x_i, y_j) = t$$

with variables $x_1, \ldots, x_m \in A_1$ and $y_1, \ldots, y_n \in A_2$ and coeffficients $a_1, \ldots, a_n \in A_1$, $b_1, \ldots, b_m \in A_2$, $\gamma_{ij} \in \mathcal{R}$ and $t \in A_T$.

Let $\vec{a}$ be the vector of $(a_1, \ldots, a_n)$; let $\vec{b}$ be the vector of $(b_1, \ldots, b_m)$; let $\vec{x}$ be the vector of $(x_1, \ldots, x_m)$; let $\vec{y}$ be the vector of $(y_1, \ldots, y_n)$; and let $\Gamma \in Mat_{m\times n}(\mathcal{R})$ be the matrix of (γ_{ij}). We have $\vec{x}\cdot\Gamma\vec{y} = \Gamma^\top\vec{x}\cdot\vec{y}$ and $\vec{x}\bullet\Gamma\vec{y} = \Gamma^\top\vec{x}\bullet\vec{y}$. So each equation can be written as $\vec{a}\cdot\vec{y} + \vec{x}\cdot\vec{b} + \vec{x}\cdot\Gamma\vec{y} = t$.

A GS statement can be viewed as a set $\{(\vec{a}_i, \vec{b}_i, \Gamma_i, t_i)\}_{i=1}^L$ over the corresponding set of bilinear groups $\{A_1^{(i)}, A_2^{(i)}, A_T^{(i)},\ f^{(i)}\}_{i=1}^L$ satisfying equations $\vec{a}_i\cdot\vec{y}_i + \vec{x}_i\cdot\vec{b}_i + \vec{x}_i\cdot\Gamma\vec{y}_i = t_i$. The witness is the set of corresponding variables $\{\vec{x}_i, \vec{y}_i\}_{i=1}^L$.

Commitment. Given keys $\vec{u_1} \in B_1^{\hat{m}}$ and $\vec{u_2} \in B_2^{\hat{n}}$, commitments of $\vec{x} \in A_1^m$ and $\vec{y} \in A_2^n$ are respectively computed as $\vec{c} := \iota_1(\vec{x}) + R\vec{u_1}$ and $\vec{d} := \iota_2(\vec{y}) + S\vec{u_2}$, where $R \leftarrow Mat_{m\times\hat{m}}(\mathcal{R})$ and $S \leftarrow Mat_{n\times\hat{n}}(\mathcal{R})$. We see that $\vec{c} \in B_1^m$ and $\vec{d} \in B_2^n$. The commitment keys can be one of two types. *Hiding* keys satisfy $\iota(A_1) \subseteq \langle\vec{u_1}\rangle$ and $\iota(A_2) \subseteq \langle\vec{u_2}\rangle$. So the commitments are perfectly hiding. *Binding* keys satisfy $p_1(\vec{u_1}) = \vec{0}$ and $p_2(\vec{u_2}) = \vec{0}$, and the maps $\iota_1 \circ p_1$ and $\iota_2 \circ p_2$ are non-trivial. If they are identity maps, then the commitments are perfectly binding.

Proof. For a statement consisting of several $(\vec{a}, \vec{b}, \Gamma, t)$ and a witness of corresponding variables $(\vec{x}, \vec{y})$, the proof includes commitments $(\vec{c}, \vec{d})$ of the variables and corresponding pairs $(\vec{\pi}, \vec{\psi})$, computed as follows. Generate $R \leftarrow Mat_{m\times\hat{m}}(\mathcal{R})$, $S \leftarrow Mat_{n\times\hat{n}}(\mathcal{R})$, $T \leftarrow Mat_{\hat{n}\times\hat{m}}(\mathcal{R})$ and $r_1, ..., r_\eta \leftarrow \mathcal{R}$. Compute $\vec{c} := \iota_1(\vec{x}) + R\vec{u_1}$; $\vec{d} := \iota_2(\vec{y}) + S\vec{u_2}$; $\vec{\pi} := R^\top\iota_2(\vec{b}) + R^\top\Gamma\iota_2(\vec{y}) + R^\top\Gamma S\vec{u_2} - T^\top\vec{u_2} + \sum_{i=1}^{\eta} r_i H_i\vec{u_2}$; and $\vec{\psi} := S^\top\ \iota_1(\vec{a}) + S^\top\Gamma^\top\iota_1(\vec{x}) + T\vec{u_1}$. Dimension of $\vec{b}$, $\vec{x}$ and $\vec{c}$ is m, dimension of $\vec{a}$, $\vec{y}$ and $\vec{d}$ is n, dimension of $\vec{\pi}$ is $\hat{m}$, and dimension of $\vec{\psi}$ is $\hat{n}$. To show that a variable of one equation is the same as another variable of the same or another equation, the same commitment is used for the variables. Verification for each equation's proof is to check $\iota_1(\vec{a})\bullet\vec{d} + \vec{c}\bullet\iota_2(\vec{b}) + \vec{c}\bullet\Gamma\vec{d} = \iota_T(t) + \vec{u_1}\bullet\vec{\pi} + \vec{\psi}\bullet\vec{u_2}$.

SXDH Instantiation. Bilinear pairing modules $Z_p, \mathbb{G}_1, \mathbb{G}_2$ and $\mathbb{G}_T$ and map e are sufficient to specify all equations in a statement. So *Para* includes setup $Gk = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, P_1, P_2)$ and CRS $\sigma = (B_1, B_2, B_T, F, \iota_1, p_1, \iota_2, p_2, \iota_1', p_1', \iota_2', p_2', \iota_T, p_T, \vec{u}, \vec{v})$ where $B_1 = \mathbb{G}_1^2$, $B_2 = \mathbb{G}_2^2$ and $B_T := \mathbb{G}_T^4$ with entry-wise group operations. $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$ can be viewed as Z_p-modules with map e. Matrices $H_1, ..., H_\eta$ are not needed. Vectors $\vec{u}$ of $u_1, u_2 \in B_1$ and $\vec{v}$ of $v_1, v_2 \in B_2$ are commitment keys for $\mathbb{G}_1$ and $\mathbb{G}_2$. More descriptions of $(\iota_1, p_1, \iota_2, p_2, \iota_1', p_1', \iota_2', p_2', \iota_T, p_T)$ are shown below.

There are 4 types of equations in statements. For *pairing product*, $A_1 = \mathbb{G}_1$, $A_2 = \mathbb{G}_2$, $A_T = \mathbb{G}_T$, $f(X, Y) = e(X, Y)$, and equations are $(\vec{A}\cdot\vec{Y})(\vec{X}\cdot\vec{B})(\vec{X}\cdot\Gamma\vec{Y}) = t_T$. For *multi-scalar multiplication* in $\mathbb{G}_1$, $A_1 = \mathbb{G}_1$, $A_2 = Z_p$, $A_T = \mathbb{G}_1$, $f(X, y) = yX$, and equations are $\vec{A}\cdot\vec{y} + \vec{X}\cdot\vec{b} + \vec{X}\cdot\Gamma\vec{y} = T_1$. For *multi-scalar multiplication* in $\mathbb{G}_2$, $A_1 = Z_p$, $A_2 = \mathbb{G}_2$, $A_T = \mathbb{G}_2$, $f(x, Y) = xY$, and equations are $\vec{a}\cdot\vec{Y} + \vec{x}\cdot\vec{B} + \vec{x}\cdot\Gamma\vec{Y} = T_2$. For *quadratic* equations, $A_1 = Z_p$, $A_2 = Z_p$, $A_T = Z_p$, $f(x, y) = xy \bmod p$ and equations are $\vec{a}\cdot\vec{y} + \vec{x}\cdot\vec{b} + \vec{x}\cdot\Gamma\vec{y} = t$.

The commitment key $\vec{u}$ contains $u_1 := (P, Q)$, where $Q = \alpha P$ with $\alpha \leftarrow Z_p*$, and $u_2 := (U, V)$ which can be computed in one of two ways. Compute $u_2 := tu_1$

for a perfectly binding key, or compute $u_2 := tu_1 - (O, P)$ for a perfectly hiding key, with $t \leftarrow Z_p*$. Under DDH, these two types of keys are computationally indistinguishable. To commit $X \in \mathbb{G}_1$, define $\iota_1(X) := (O, X)$ and compute the commitment $c := \iota_1(X) + r_1u_1 + r_2u_2$ with $r_1, r_2 \leftarrow Z_p$. Define $p_1 : (C_1, C_2) \mapsto C_2 - \alpha C_1$, then the commitment is perfectly binding for the binding key, and the commitment is perfectly hiding for the hiding key. To commit $x \in Z_p$ in $\mathbb{G}_1$, define $u = u_2 + (O, P)$, $\iota'_1(x) := xu$ and $p'_1(c_1P, c_2P) := c_2 - \alpha c_1$, and compute the commitment $c' := \iota'_1(x) + r'u_1$ with $r' \leftarrow Z_p$. Committing $Y \in \mathbb{G}_2$ and $y \in Z_p$ in $\mathbb{G}_2$ is similarly defined with $\vec{v}$ and maps $\iota_2, p_2, \iota'_2, p'_2$.

A proof and its verification can then be done as specified in the general GS proofs. GS proofs are WI and in some cases ZK. As shown in [2], in the SXDH and Decisional Linear (DLIN) [28] instantiations, for statements consisting of only multi-scalar multiplication and quadratic equations, GS proofs are composable ZK.

8.3 Accumulator

An *universal accumulator* [9, 10] consists of the following PPT algorithms.

- Setup takes in 1^l and outputs $(Para, Aux)$, where $Para$ is setup parameters containing a domain Dom_{Para} of elements to be accumulated and Aux is some *auxiliary information*.
- Accu takes in $Para$ and a set of elements $AcSet$ and returns an accumulator value $AcVal$. In some cases, Accu may also take in Aux to compute $AcVal$ more efficiently. The input as a set, where order makes no difference, instead of a sequence implies the quasi commutativity property defined in previous papers [6, 7].
- A membership proof system (Setup, ProveMem, VerifyMem) proves that an element Ele is accumulated in $AcVal$. Note that $AcSet$ is not an input. There is a PPT algorithm CompMemWit to compute a *membership witness* for this proof from $Para$, Ele, $AcSet$ and $AcVal$.
- An **non-membership (NM)** proof system (Setup, ProveNM, VerifyNM) proves that an element Ele is **not** accumulated in $AcVal$. Note that $AcSet$ is not an input. There is a PPT algorithm CompNMWit to compute an *NM witness* for this proof from $Para$, Ele, $AcSet$ and $AcVal$.

An accumulator is *dynamic* if there exist the following 3 PPT algorithms, whose costs should not depend on $AcSet$'s size, for adding or removing an accumulated element Ele. UpdateVal, whose input includes $Para$, Ele, the current accumulator value $AcVal$ and Aux, updates the accumulator value. UpdateMemWit, whose input includes $Para$, Ele, the current witness Wit and $AcVal$, updates membership witnesses. For universal accumulators, UpdateNMWit, whose input includes $Para$, Ele, the current witness Wit and $AcVal$, updates NM witnesses.

Security of accumulators is implied by completeness and soundness of the 2 proof systems.

Note that this paper will ignore membership proofs and only focus on non-membership proofs.

8.4 BCCKLS Delegatable Anonymous Credentials without Delegatable Revocation

In a delegatable anonymous credential system [1], each user U has a secret key Sk_U. For each transaction with another user V, U uses a new pseudonym $Nym_U^{(V)}$, which is generated from Sk_U but reveals nothing about Sk_U. A user can become a root authority of credentials by publishing one of her pseudonyms. A user can prove to a verifier that she possesses a valid credential which is delegated through a sequence of users starting from the root authority. That sequence is the user's *chain of delegators*.

A delegatable anonymous credential system consists of the following algorithms. Setup generates the public setup parameters. KeyGen outputs a secret key for a user. NymGen produces a new pseudonym with some auxiliary information. Protocol Issue $\leftrightarrow$ Obtain allows an issuer to issue a credential, which is delegated through a number of levels from a root authority, to a user. CredProve produces a proof of possessing such a credential. CredVerify verifies if the credential proof is valid.

An delegatable anonymous credential system has three security requirements. Correctness means that an honest user always gets a valid credential from a honest issuer and can use it to generate a valid credential proof, which is always accepted by a verifier. Anonymity means that no adversary can obtain or link any information about an honest user's identity and credential delegation from interacting with the system's algorithms. Unforgeability means that no adversary can forge a proof of possessing a credential which is delegated through a chain of honest users.

9 Security Proofs for the Homomorphic Proof and ADNMP constructions

9.1 Proof of theorem 1

We need to prove that $(\Pi_{GS}, +_{GS}, I_{GS})$ satisfies the 5 conditions of an abelian group.
Closure: We can see that $(Sta, Wit, Proof) \leftarrow (Sta_1, Wit_1, Proof_1) +_{GS} (Sta_2, Wit_2, Proof_2)$ (as in the description) satisfies the requirements for an element in Π_{GS} as follows. $\forall i \in M$: $x[i] = x_1[i] = x_0[i]$ and $c[i] = c_1[i] = c_0[i]$. $\forall j \in \bar{M}$: $b[j] = b_1[j] = b_0[j]$. $\forall i \in N$: $y[i] = y_1[i] = y_0[i]$ and $d[i] = d_1[i] = d_0[i]$. $\forall j \in \bar{N}$: $a[j] = a_1[j] = a_0[j]$. If $(i \in \bar{M}) \vee (j \in \bar{N})$, then $\Gamma[i,j] = \Gamma_1[i,j] = \Gamma_0[i,j]$. We now need to prove that *Proof* is the valid proof of *Sta* and *Wit*. Suppose for $i \in \{1,2\}$, $\vec{c_i} := \iota_1(\vec{x_i}) + R_i\vec{u_1}$, $\vec{d_i} := \iota_2(\vec{y_i}) + S_i\vec{u_2}$.

$$\begin{aligned}\vec{\pi}_i &:= R_i^\top \iota_2(\vec{b_i}) + R_i^\top \Gamma_i \iota_2(\vec{y_i}) + R_i^\top \Gamma_i S_i \vec{u_2} \\ &\quad - T_i^\top \vec{u_2} + \sum_{j=1}^{\eta} r_j^{(i)} H_j \vec{u_2}\end{aligned} \tag{1}$$

$$\vec{\psi}_i := S_i^\top \iota_1(\vec{a_i}) + S_i^\top \Gamma_i^\top \iota_1(\vec{x_i}) + T_i \vec{u_1} \tag{2}$$

Without losing generality, for $i \in \{1,2\}$, we can write

$$\vec{x_i} := \begin{pmatrix} \hat{X} \\ \tilde{X}_i \end{pmatrix}, \vec{b_i} := \begin{pmatrix} \hat{B}_i \\ \tilde{B} \end{pmatrix}, R_i := \begin{pmatrix} \hat{R} \\ \tilde{R}_i \end{pmatrix}, \vec{c_i} := \begin{pmatrix} \hat{C} \\ \tilde{C}_i \end{pmatrix}$$

where $\hat{X}$ consists of $x[j]$ with $j \in M$ and $\tilde{X}_i$ consists of $x_i[j]$ with $j \in \bar{M}$; $\hat{B}_i$ consists of $b_i[j]$ with $j \in M$ and $\tilde{B}$ consists of $b[j]$ with $j \in \bar{M}$; and $\hat{R}$ consists of rows j of R_i with $j \in M$ and $\tilde{R}_i$ consists of rows j of R_i with $j \in \bar{M}$; and $\hat{C}$ consists of $c[j]$ with $j \in M$ and $\tilde{C}_i$ consists of $C_i[j]$ with $j \in \bar{M}$. Now we have

$$\begin{aligned}
\vec{x} &= \begin{pmatrix} \hat{X} \\ \tilde{X}_1 + \tilde{X}_2 \end{pmatrix}, \ \vec{b} = \begin{pmatrix} \hat{B}_1 + \hat{B}_2 \\ \tilde{B} \end{pmatrix}, \ R = \begin{pmatrix} \hat{R} \\ \tilde{R}_1 + \tilde{R}_2 \end{pmatrix} \\
\vec{c} &= \begin{pmatrix} \hat{C} \\ \tilde{C}_1 + \tilde{C}_2 \end{pmatrix} = \begin{pmatrix} \iota_1(\hat{X}) + \hat{R}\vec{u_1} \\ \iota_1(\tilde{X}_1) + \tilde{R}_1\vec{u_1} + \iota_1(\tilde{X}_2) + \tilde{R}_2\vec{u_1} \end{pmatrix} \\
&= \begin{pmatrix} \iota_1(\hat{X}) + \hat{R}\vec{u_1} \\ \iota_1(\tilde{X}_1 + \tilde{X}_2) + (\tilde{R}_1 + \tilde{R}_2)\vec{u_1} \end{pmatrix} = \iota_1(\vec{x}) + R\vec{u_1} \qquad (3)
\end{aligned}$$

which is how commitment $\vec{c}$ should be generated from $\vec{x}$ and R for the proof. In the same way, without losing generality, for $i \in \{1,2\}$, we can write

$$\vec{y_i} := \begin{pmatrix} \hat{Y} \\ \tilde{Y}_i \end{pmatrix}, \vec{a_i} := \begin{pmatrix} \hat{A}_i \\ \tilde{A} \end{pmatrix}, S_i := \begin{pmatrix} \hat{S} \\ \tilde{S}_i \end{pmatrix}, \vec{d_i} := \begin{pmatrix} \hat{D} \\ \tilde{D}_i \end{pmatrix}$$

where $\hat{Y}$ consists of $y[j]$ with $j \in N$ and $\tilde{Y}_i$ consists of $y_i[j]$ with $j \in \bar{N}$; $\hat{A}_i$ consists of $a_i[j]$ with $j \in N$ and $\tilde{A}$ consists of $a[j]$ with $j \in \bar{N}$; $\hat{S}$ consists of rows j of S_i with $j \in N$ and $\tilde{S}_i$ consists of rows j of S_i with $j \in \bar{N}$; and $\hat{D}$ consists of $d[j]$ with $j \in N$ and $\tilde{D}_i$ consists of $D_i[j]$ with $j \in \bar{N}$. Now we have

$$\begin{aligned}
\vec{y} &= \begin{pmatrix} \hat{Y} \\ \tilde{Y}_1 + \tilde{Y}_2 \end{pmatrix}, \vec{a} = \begin{pmatrix} \hat{A}_1 + \hat{A}_2 \\ \tilde{A} \end{pmatrix}, S = \begin{pmatrix} \hat{S} \\ \tilde{S}_1 + \tilde{S}_2 \end{pmatrix} \\
\vec{d} &= \begin{pmatrix} \hat{D} \\ \tilde{D}_1 + \tilde{D}_2 \end{pmatrix} = \iota_2(\vec{y}) + S\vec{u_2} \qquad (4)
\end{aligned}$$

showing how commitment $\vec{d}$ is generated from $\vec{y}$ and S for the proof. Besides, we have for $i \in \{1,2\}$

$$\Gamma_i := \begin{pmatrix} \hat{\Gamma}_i & \check{\Gamma} \\ \tilde{\Gamma} & O \end{pmatrix}, \ \Gamma := \begin{pmatrix} \hat{\Gamma}_1 + \hat{\Gamma}_2 & \check{\Gamma} \\ \tilde{\Gamma} & O \end{pmatrix} \qquad (5)$$

where $\hat{\Gamma}_i$ consists of $\Gamma[j,k]$ with $j \in M$ and $k \in N$, $\check{\Gamma}$ consists of $\Gamma[j,k]$ with $j \in M$ and $k \in \bar{N}$, $\tilde{\Gamma}$ consists of $\Gamma[j,k]$ with $j \in \bar{M}$ and $k \in N$, and a zero matrix of $\Gamma[j,k]$ with $j \in \bar{M}$ and $k \in \bar{N}$. Substituting (3) and (5) in (1) and (2), we write $\pi = \pi_1 + \pi_2$

$$\begin{aligned}
\vec{\pi} = &\Bigg(\begin{pmatrix} \hat{R}^\top & \tilde{R}_1^\top \end{pmatrix} \begin{pmatrix} \iota_2(\hat{B}_1) \\ \iota_2(\tilde{B}) \end{pmatrix} + \begin{pmatrix} \hat{R}^\top & \tilde{R}_2^\top \end{pmatrix} \begin{pmatrix} \iota_2(\hat{B}_2) \\ \iota_2(\tilde{B}) \end{pmatrix} \Bigg) \\
&+ \Bigg(\begin{pmatrix} \hat{R}^\top & \tilde{R}_1^\top \end{pmatrix} \begin{pmatrix} \hat{\Gamma}_1 & \check{\Gamma} \\ \tilde{\Gamma} & O \end{pmatrix} \begin{pmatrix} \iota_2(\hat{Y}) \\ \iota_2(\tilde{Y}_1) \end{pmatrix} \\
&+ \begin{pmatrix} \hat{R}^\top & \tilde{R}_2^\top \end{pmatrix} \begin{pmatrix} \hat{\Gamma}_2 & \check{\Gamma} \\ \tilde{\Gamma} & O \end{pmatrix} \begin{pmatrix} \iota_2(\hat{Y}) \\ \iota_2(\tilde{Y}_2) \end{pmatrix} \Bigg) \\
&+ \Bigg(\begin{pmatrix} \hat{R}^\top & \tilde{R}_1^\top \end{pmatrix} \begin{pmatrix} \hat{\Gamma}_1 & \check{\Gamma} \\ \tilde{\Gamma} & O \end{pmatrix} \begin{pmatrix} \hat{S} \\ \tilde{S}_1 \end{pmatrix} \\
&+ \begin{pmatrix} \hat{R}^\top & \tilde{R}_2^\top \end{pmatrix} \begin{pmatrix} \hat{\Gamma}_2 & \check{\Gamma} \\ \tilde{\Gamma} & O \end{pmatrix} \begin{pmatrix} \hat{S} \\ \tilde{S}_2 \end{pmatrix} \Bigg) \vec{u_2} \\
&- (T_1^\top + T_2^\top)\vec{u_2} + (\sum_{j=1}^{\eta} r_j^{(1)} H_j + \sum_{j=1}^{\eta} r_j^{(2)} H_j)\vec{u_2}
\end{aligned}$$

Multiplying matrices and regrouping with (3) and (5) yields

$$\begin{aligned}
\vec{\pi} = &(\begin{pmatrix} \hat{R}^\top & (\tilde{R}_1 + \tilde{R}_2)^\top \end{pmatrix} \begin{pmatrix} \iota_2(\hat{B}_1 + \hat{B}_2) \\ \iota_2(\tilde{B}) \end{pmatrix} \\
&+ \begin{pmatrix} \hat{R}^\top(\hat{\Gamma}_1 + \hat{\Gamma}_2) + (\tilde{R}_1 + \tilde{R}_2)^\top \tilde{\Gamma} & \hat{R}^\top \check{\Gamma} \end{pmatrix} \begin{pmatrix} \iota_2(\hat{Y}) \\ \iota_2(\tilde{Y}_1 + \tilde{Y}_2) \end{pmatrix} \\
&+ \begin{pmatrix} \hat{R}^\top(\hat{\Gamma}_1 + \hat{\Gamma}_2) + (\tilde{R}_1 + \tilde{R}_2)^\top \tilde{\Gamma} & \hat{R}^\top \check{\Gamma} \end{pmatrix} \begin{pmatrix} \hat{S} \\ \tilde{S}_1 + \tilde{S}_2 \end{pmatrix} \vec{u_2} \\
&- T^\top \vec{u_2} + \sum_{j=1}^{\eta} r_j H_j \vec{u_2}
\end{aligned}$$

Replacing $\vec{b}$ and R from (3) and $\vec{y}$ and S from (4), we have

$$\vec{\pi} = R^\top \iota_2(\vec{b}) + R^\top \Gamma \iota_2(\vec{y}) + R^\top \Gamma S \vec{u_2} - T^\top \vec{u_2} + \sum_{j=1}^{\eta} r_j H_j \vec{u_2}$$

Similarly, we can show that $\vec{\psi} := S^\top \iota_1(\vec{a}) + S^\top \Gamma^\top \iota_1(\vec{x}) + T\vec{u_1}$. So $\vec{c}$, $\vec{d}$, $\vec{\pi}$, and $\vec{\psi}$ are generated according to the formula for a GS proof of $(\vec{a}, \vec{b}, \Gamma, t)$ and $(\vec{x}, \vec{y})$. Therefore, *Proof* is a valid proof of *Sta* and *Wit*.

It is straightforward to validate the other 4 conditions **Associativity**, **Commutativity**, **Identity element** and **Inverse element** of abelian groups. So the theorem holds.

9.2 Proof of theorem 2

We provide the theorem proof in the GS SXDH instantiation. The theorem proof in the GS SDLIN (Symmetric DLIN) [28] instantiation is similar.

The proof system's *completeness* comes from completeness of the GS proof system and the fact that $y_2 \notin AcSet$ and $X_{j2} \neq 0$ means $T_j \neq 0, \forall j \in \{1, \dots, m\}$.

The proof system's GS statement consists of only multi-scalar equations. So as explained in appendix 8.2, if we use the GS SXDH instantiation, the NM proof system for this accumulator is *composable ZK*. As described in [2], we can simulate a setup and a proof which are respectively computationally indistinguishable from a real setup and a real proof generated from the simulated setup.

Now we prove *soundness* of the NM proof system. Suppose a PPT adversary Adv can forge an NM proof that VerifyNM accepts for equations $\bigwedge_{j=1}^{m}((y_1 + y_2)X_{j1} + y_{j3}P_1 = V_j \wedge\ X_{j3} - y_{j3}A = 0 \wedge\ y_{j3}X_{j2} = T_j)$ where $T_j \neq 0$ but y_2 is accumulated in one of V_js with non-negligible probability. We show how to use it to break ESDH.

Suppose we are given the assumption challenge $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, P_1, \delta P_1, \dots, \delta^{q+1}P_1, A, P_2, \delta P_2)$. As the commitment keys are perfectly binding, we are able to simulate the accumulator's setup parameter $Para_{sim}$ with extracting trapdoor so that from a commitment in $\mathbb{G}_2$ of $y \in Z_p$ and a commitment of $X \in \mathbb{G}_1$, we can respectively extract yP_2 and X, as follows.

Simulated $Para_{sim}$ includes setup $Gk = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T,\ e, P_1, P_2)$, A, CRS $\sigma = (B_1, B_2, B_T, F, \iota_1, p_1, \iota_2, p_2, \iota_1', p_1', \iota_2', p_2', \iota_T, p_T, \vec{u}, \vec{v})$, $\varsigma = (P_1, \delta P_1, \dots, \delta^{q+1}P_1)$ and τ. $(B_1, B_2, B_T, F, \iota_1, p_1, \iota_2, p_2, \iota_1', p_1', \iota_2', p_2', \iota_T, p_T)$ is defined as in the normal SXDH instantiation. Choose $P_u \leftarrow \mathbb{G}_1$ and $\alpha_u, t_u \leftarrow Z_p*$, compute $Q_u = \alpha_u P_u$, $u_1 := (P_u, Q_u)$ and $u_2 := t_u u_1$, and simulate $\vec{u} = (u_1, u_2)$. Choose $\alpha_v, \beta_v, t_v \leftarrow Z_p*$, compute $P_v = \beta_v P_2$, $Q_v = \alpha_v P_v$, $v_1 := (P_v, Q_v)$ and $v_2 := t_v v_1$, and simulate $\vec{v} = (v_1, v_2)$. The trapdoor is $(\alpha_u, t_u, \alpha_v, \beta_v, t_v)$. With the trapdoor and δP_2 , we can compute $\tau = \iota_2'(\delta) = \delta(v_2 + (0, P_v))$. So we have simulated the accumulator's setup parameter which is indistinguishable from a real setup parameter.

Furthermore, with the knowledge of α_u, t_u, given a commitment $c = (C_1, C_2) = (O, X) + r_1 u_1 + r_2 u_2$ of $X \in \mathbb{G}_1$, we can extract $X = C_2 - \alpha_u C_1$. With the knowledge of α_v, β_v, t_v, given a $\mathbb{G}_2$ commitment $c = (C_1, C_2) = y(v_2 + (0, P_v)) + r'v_1$ of $y \in Z_p$, we can extract $yP_2 = (C_2 - \alpha_v C_1)^{1/\beta_v}$.

Now we provide the PPT adversary Adv the simulated $Para_{sim}$, and Adv can forge the NM proof with non-negligible probability. The soundness of GS proof system ensures that the GS equations hold $\bigwedge_{j=1}^{m}((y_1 + y_2)X_{j1} + y_{j3}P_1 = V_j \wedge X_{j3} - y_{j3}A = 0 \wedge\ y_{j3}X_{j2} = T_j)$.

The forged proof contains commitments of X_{j1}, X_{j3}, X_{j2} and of $y_1 = \delta, y_2, y_{j3}$ in $\mathbb{G}_2$. So we can extract X_{j1}, X_{j3}, X_{j2} and $y_2P_2, y_{j3}P_2$ and know $y_{j3} \neq 0$. As y_2 is in $AcSet$, we can find y_2. Suppose y_2 is accumulated in V_l which accumulates $\{a_1, ..., a_k\}$. As $X_{l3} = y_{l3}A$, we can extract X_{l1}, y_2 and $(y_{l3}P_2, y_{l3}A)$. We have $(y_1 + y_2)X_{l1} + y_{l3}P_1 = \prod_{i=1}^{k}(y_1 + a_i)y_1P_1$ and $y_2 \in \{a_1, ..., a_k\}$, so we can compute $\frac{y_{l3}}{y_1+y_2}P_1$ from X_{l1}, $\{a_1, ..., a_k\}$ and ς. So now, we can find $(\frac{y_{l3}}{\delta+y_2}P_1, y_2, y_{l3}P_2, y_{l3}A)$ and break the ESDH assumption.

9.3 Proof of theorem 3

Proving Delegability. In $\mathsf{CompNMProof}(Para, De, AcSet, AcVal)$, after dividing De into proofs $NMProof_i$ of $(\delta+y_2)X_1^{(i)}+y_3^{(i)}P_1 = \delta^i P_1 \wedge X_3^{(i)} - y_3^{(i)}A = 0 \wedge y_3^{(i)}X_2 = T^{(i)}$ for $i \in \{1, ..., q+1\}$, $NMProof_i$s form to a set of homomorphic proofs. For each component accumulator value $V = \prod_{i=1}^{k}(\delta + a_i)\delta P_1$ of $\{a_1, ..., a_k\}$, let $b_0 = 1$ and $b_i = \sum_{1 \le j_1 < j_2 < ... < j_i \le k} \prod_{l=1}^{i} a_{j_l}$, for $i \in \{0, ..., k\}$. So the computed $NMProof = \sum_{i=0}^{k} b_i NMProof_{k+1-i}$ is a proof of $(\delta + y_2)X_1 + y_3P_1 = V \wedge X_3 - y_3A = 0 \wedge y_3X_2 = T$, where $X_1 = \sum_{i=0}^{k} b_i X_1^{(k+1-i)}$, $X_3 = \sum_{i=0}^{k} b_i X_3^{(k+1-i)}$, $y_3 = \sum_{i=0}^{k} b_i y_3^{(k+1-i)}$ and $T = \sum_{i=0}^{k} b_i T^{(k+1-i)}$. This is a non-membership proof that y_2 is not accumulated in the component accumulator value V.

Replacing T by T' and $SubProof$ by $SubProof'$ in $NMProof$ results in a new proof $NMProof'$ of $(\delta + y_2)X_1 + y_3P_1 = V \wedge X_3 - y_3A = 0 \wedge y_3X_2' = T'$, which is also a non-membership proof that y_2 is not accumulated in the component accumulator value V.

Concatenating these $NMProof'$ of all V_j in $AcVal$ and randomizing the concatenation produce a randomized proof of equations $\bigwedge_{j=1}^{m}((y_1 + y_2)X_{j1} + y_{j3}P_1 = V_j \wedge X_{j3} - y_{j3}A = 0 \wedge y_{j3}X_{j2} = T_j)$ which are the same as equations for the proof outputted by $\mathsf{ProveNM}$. Due to GS proofs' randomizability, these $\mathsf{CompNMProof}$'s and $\mathsf{ProveNM}$'s outputs have the same distribution, which means Delegability.

Proving Unlinkability. We prove that if an adversary can break the accumulator's Unlinkability, then we can break either q-DSDH or GS's underlying assumption (SXDH or SDLIN). There are 2 cases.

If the adversary can distinguish between a GS proof De and its simulated proof both in a simulated setup with non-negligible probability, then due to the GS proof system's composable ZK, we can break the underlying assumption.

Otherwise, we can break q-DSDH as follows. Suppose we are given a q-DSDH challenge $p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, P_1, P_2, B_0, x_0B_0, \ldots, x_0^qB_0, B_1, x_bB_1, \ldots, x_b^qB_1$. Generate a CRS for GS proofs from the GS setup $Gk = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, P_1, P_2)$ as in the normal GS setup algorithm. Using the same simulation for GS proofs [2], simulate a proof De for $\bigwedge_{i=1}^{q+1}((\delta + y_2)X_1^{(i)} + y_3^{(i)}P_1 = \delta^iP_1 \wedge X_3^{(i)} - y_3^{(i)}A = 0 \wedge y_3^{(i)}X_2 = (-1)^i x_0^{i-1}B_0)$, and a proof De_b for $\bigwedge_{i=1}^{q+1}((\delta + y_2)X_1^{(i)} + y_3^{(i)}P_1 = \delta^iP_1 \wedge X_3^{(i)} - y_3^{(i)}A = 0 \wedge y_3^{(i)}X_2 = (-1)^i x_b^{i-1}B_1)$. We then give the adversary De and De_b. As the adversary can not distinguish between a delegating key and a simulated one with non-negligible probability, and he can break Unlinkability, he can tell with non-negligible advantage over a random guess if b is 0 or 1. That breaks q-DSDH.

Proving Redelegability. In $\mathsf{Rede}(Para, De)$, for each proof $Proof_i$ of $y_3^{(i)}X_2 = T^{(i)}$ in De $(i \in \{1, ..., q+1\})$, compute $Proof_i' = rProof_i$ which is a proof of

$y_3^{(i)} X_2' = T'^{(i)}$, where $r \leftarrow Z_p^*$, $X_2' = rX_2$ and $T'^{(i)} = rT^{(i)}$. We see that for the same y_2, the output $T'^{(i)}$ of Rede has the same distribution as the output $T^{(i)}$ of Dele, for $i \in \{1, ..., q+1\}$.

Additionally, for the same $T^{(i)}$, $i \in \{1, ..., q+1\}$, Rede's output is a randomization of the GS proof that Dele can produce for GS equations $\bigwedge_{i=1}^{q+1}((\delta + y_2)X_1^{(i)} + y_3^{(i)} P_1 = \delta^i P_1 \wedge X_3^{(i)} - y_3^{(i)} A = 0 \wedge y_3^{(i)} X_2 = T^{(i)})$. Therefore, Dele and Rede output the same distribution that leads to Redelegability.

Proving Verifiability. Verifiability comes from the completeness and soundness of GS proofs, as De is a GS proof.

10 RDAC Security Definitions

This section presents how to extend the security definitions of delegatable anonymous credentials [1] to provide security definitions of Revocable Delegatable Anonymous Credential systems. We provide the definitions in the more general case, where a user must prove that her credential and all of its ancestors are not revoked. The other case, where a user only has to prove that her credential is not revoked, is simpler and can be derived from this.

A pair of credential $Cred$ and delegation information $DeInf$ is a 'proper level L' pair for Nym_O with respect to $(Para_{DC}, Sk, Rn)$ and a blacklist BL if and only if proofs, that are generated from the pair input to $\mathsf{CredProve}$, are always accepted for all nyms with ancestors not blacklisted in BL. Formally, the pair $(Cred, DeInf)$ satisfies the following.

$$\begin{aligned} &Pr[(Nym, Aux(Nym)) \leftarrow \mathsf{NymGen}(Para_{DC}, Sk, Rn);\\ &\quad CredProof \leftarrow \mathsf{CredProve}(Para_{DC}, Nym_O, Cred, DeInf, Sk, Rn,\\ &\qquad Nym, Aux(Nym), BL, L);\\ &\quad \mathsf{CredVerify}(Para_{DC}, Nym_O, CredProof, Nym, BL, L) = accept] = 1. \end{aligned}$$

The pair predicate, denoted by $properPair(Para_{DC}, Cred, DeInf, Sk, Rn, Nym_O, BL, L)$, is $true$ if only if $Cred$ and $DeInf$ form a 'proper level L' pair. Similarly, we let $validAux(Para_{DC}, Nym, Sk, Rn, Aux(Nym))$ denote a predicate for a valid $(Nym, Aux(Nym))$ with respect to $(Para_{DC}, Sk, Rn)$, and let $Check(Para_{DC}, Nym_O, Cred, DeInf, Sk, Rn, Nym, Aux(Nym), BL, L) = validAux(Para_{DC}, Nym, Sk, Rn, Aux(Nym)) \wedge properPair(Para_{DC}, Cred, DeInf, Sk, Rn, Nym_O, BL, L)$.

10.1 Correctness

Intuitively, suppose all participants are honest. A user always gets valid credentials from issuers. If the user is not revoked, she can always generate a credential proof, which is always accepted by a verifier who does not require the user's whole credential chain not revoked. If the user's whole credential chain is not revoked, she can always generate a credential proof, which is always accepted.

The correctness requirements include:

1. If $(Nym, Aux(Nym)) \leftarrow \mathsf{NymGen}(Para_{DC}, Sk, Rn)$, then $validAux(Para_{DC}, Nym, Sk, Rn, Aux(Nym))$ is always $true$.
2. If $properPair(Para_{DC}, Cred, DeInf, Sk_I, Rn_I, Nym_O, BL, L) = false$, or if $validAux(Para_{DC}, Nym_I, Sk_I, Rn_I, Aux(Nym_I)) = false$, or if $validAux(Para_{DC}, Nym_U, Sk_U, Rn_U, Aux(Nym_U)) = false \; \forall \; (Sk_U, Rn_U, Aux(Nym_U))$; then $\mathsf{Issue}(Para_{DC}, Nym_O, Sk_I, Rn_I, Nym_I, Aux(Nym_I), Cred, DeInf, Nym_U, BL, L)$ aborts without interacting with Obtain.
3. Obtain always either aborts or outputs a credential and delegation information, which form a 'proper level $L+1$' pair with regard to a blacklist BL.
4. Users with 'improper' pairs will abort whereas users with a 'proper level L' pair can delegate 'proper level $L+1$' pairs.
5. If $properPair(Para_{DC}, Cred, DeInf, Sk, Rn, Nym_O, BL, L) = false$, or if $validAux(Para_{DC}, Nym, Sk, Rn, Aux(Nym)) = false$; then $\mathsf{CredProve}(Para_{DC}, Nym_O, Cred, DeInf, Sk, Rn, Nym, Aux(Nym), BL, L)$ aborts.

10.2 Anonymity

It means an adversary, who could collude some participants in the system, can not gain any information about honest participants. The anonymity definition requires that the adversary's interaction with honest parties is indistinguishable from interaction with a simulator, whose algorithms include SimSetup, SimProve, SimObtain and SimIssue. The additions to the definition in [1] include the followings. Nym reveals no information about its r-nym. New entities r-nyms, blacklist and delegation information could be generated as part of challenges by the adversary to the simulator.

Formally, it requires that there exists a simulator SimSetup, SimProve, SimObtain, SimIssue such that, for a user Nym with corresponding $(Sk, Rn, Aux(Nym))$, the simulator without any knowledge of $(Sk, Rn, Aux(Nym))$ can simulate the user's execution of the system's protocols, and the simulation is indistinguishable from the real execution. Formally, the simulator must satisfy the following conditions.

1. The output of SimSetup is indistinguishable from those generated by Setup. $|Pr[(Para_{DC}, Sk_{BA}, BL_e) \leftarrow \mathsf{Setup}(1^k); b \leftarrow A(Para_{DC}, Sk_{BA}) : b = 1] - Pr[(Para_{DC}, Sk_{BA}, BL_e, sim) \leftarrow \mathsf{SimSetup}(1^k); b \leftarrow A(Para_{DC}, Sk_{BA}) : b = 1]|$ is negligible.
2. Consider $(Para_{DC}, Sk_{BA}, BL_e, sim) \leftarrow \mathsf{SimSetup}(1^k)$; $(Sk, Rn) \leftarrow \mathsf{KeyGen}(Para_{DC})$; and $(Nym, Aux(Nym)) \leftarrow \mathsf{NymGen}(Para_{DC}, Sk, Rn)$. Then from $(Para_{DC}, Sk_{BA}, sim, Nym)$, a PPT adversary gains no information about (Sk, Rn).

3. SimProve can simulate a credential proof indistinguishable from a real one, even without knowledge of Sk, Rn, $Cred$ and $DeInf$, which are chosen by the adversary. That means $\forall$ PPT adversaries $A = (A_1, A2)$: $|Pr[(Para_{DC}, Sk_{BA}, BL_e, sim) \leftarrow \mathsf{SimSetup}(1^k); (Nym_O, Cred, DeInf, Sk, Rn, Nym, Aux(Nym), BL, L, state) \leftarrow A_1(Para_{DC}, Sk_{BA}, sim); CredProof \leftarrow \mathsf{CredProve}(Para_{DC}, Nym_O, Cred, DeInf, Sk, Rn, Nym, Aux(Nym), BL, L); b \leftarrow A_2(state, CredProof) : b = 1] - Pr[(Para_{DC}, Sk_{BA}, BL_e, sim) \leftarrow \mathsf{SimSetup}(1^k); (Nym_O, Cred, DeInf, Sk, Rn, Nym, Aux(Nym), BL, L, state) \leftarrow A_1(Para_{DC}, Sk_{BA}, sim); flag \leftarrow Check(Para_{DC}, Nym_O, Cred, DeInf, Sk, Rn, Nym, Aux(Nym), BL, L); CredProof \leftarrow \mathsf{SimProve}(Para_{DC}, sim, Nym_O, Nym, BL, L, flag); b \leftarrow A_2(state, CredProof) : b = 1]|$ is negligible.
4. SimObtain can simulate interactions indistinguishable from interactions by Obtain. That means $\forall$ PPT adversaries $A = (A_1, A2)$: $|Pr[(Para_{DC}, Sk_{BA}, BL_e, sim) \leftarrow \mathsf{SimSetup}(1^k); (Nym_O, Sk, Rn, Nym, Aux(Nym), BL, L, Nym_A, state) \leftarrow A_1(Para_{DC}, Sk_{BA}, sim); b \leftarrow A_2(state) \leftrightarrow \mathsf{Obtain}(Para_{DC}, Nym_O, Sk, Rn, Nym, Aux(Nym), Nym_A, BL, L) : b = 1] - Pr[(Para_{DC}, Sk_{BA}, BL_e, sim) \leftarrow \mathsf{SimSetup}(1^k); (Nym_O, Sk, Rn, Nym, Aux(Nym), BL, L, Nym_A, state) \leftarrow A_1(Para_{DC}, Sk_{BA}, sim); flag \leftarrow (validAux(Para_{DC}, Nym, Sk, Rn, Aux(Nym)) \wedge (Rn \notin BL)); b \leftarrow A_2(state) \leftrightarrow \mathsf{SimObtain}(Para_{DC}, sim, Nym_O, Nym, Nym_A, BL, L, flag) : b = 1]|$ is negligible.
5. One major difference with the anonymity definition in [1] is this property. It requires that SimIssue can simulate interactions indistinguishable from interactions by Issue. However, r-nyms on the chain of issuer's credentials are randomly generated and not revealed to the adversary, because as discussed before, a user and BA can tell if a given r-nym belongs to one of the delegators on her chain. Formally, $\forall$ PPT adversaries $A = (A_1, A_2)$: $|Pr[(Para_{DC}, Sk_{BA}, BL_e, sim) \leftarrow \mathsf{SimSetup}(1^k); (Nym_O, Sk, Nym, Aux(Nym), Cred, BL, L, Nym_A, state) \leftarrow A_1(Para_{DC}, Sk_{BA}, sim); (Rn, DeInf) \leftarrow S_{DI}(Para_{DC}, Nym_O, Cred, Sk, Nym, Aux(Nym), BL, L); \mathsf{Issue}(Para_{DC}, Nym_O, Sk, Rn, Nym, Aux(Nym), Cred, DeInf, Nym_A, BL, L) \leftrightarrow A_2(state) \to b : b = 1] - Pr[(Para_{DC}, Sk_{BA}, BL_e, sim) \leftarrow \mathsf{SimSetup}(1^k); (Nym_O, Sk, Nym, Aux(Nym), Cred, BL, L, Nym_A, state) \leftarrow A_1(Para_{DC}, Sk_{BA}, sim); (Rn, DeInf) \leftarrow S_{DI}(Para_{DC}, Nym_O, Cred, Sk, Nym, Aux(Nym), BL, L); flag \leftarrow Check(Para_{DC}, Nym_O, Cred, DeInf, Sk, Rn, Nym, Aux(Nym), BL, L); \mathsf{SimIssue}(Para_{DC}, sim, Nym_O, Nym, Nym_A, BL, L, flag) \leftrightarrow A_2(state) \to b : b = 1]|$ is negligible.
   Denote $S_{DI}(Para_{DC}, Nym_O, Cred, Sk, Nym, Aux(Nym), BL, L)$ the distribution of $(Rn, DeInf)$ such that $Check(Para_{DC}, Nym_O, Cred, DeInf, Sk, Rn, Nym, Aux(Nym), BL, L)$ outputs $true$. If $S_{DI}(Para_{DC}, Nym_O, Cred, Sk, Nym, Aux(Nym), BL, L)$ is empty, then $\forall (Rn, DeInf), Check(Para_{DC}, Nym_O, Cred, DeInf, Sk, Rn, Nym, Aux(Nym), BL, L)$ returns $false$.

10.3 Unforgeability

Briefly, it means that an adversary, who could interact with the system in many ways, could not forge a valid credential proof for a challenge Nym of an r-nym and a secret key, which are in one of rogue conditions. It also assumes complete binding of Nyms, so that exactly one r-nym and one key could be extracted from a Nym. The adversary's interaction with the system is modelled by an Oracle, who could perform several tasks based on the adversary's request. The additions to the definition in [1] include the followings. Apart from the condition that there is no chain of honest users who delegate the challenge Nym, another rogue condition is that the challenge r-nym is blacklisted by an honest BA. If a credential proof is required to prove that all users on its chain are not revoked, another rogue condition is that a user on the challenge Nym's credential chain is blacklisted by an honest BA.

Formally, similar to [1], assume F is an efficiently computable bijection, it requires the existence of an extractor ($\mathsf{ExtSetup}, \mathsf{Extract}$) satisfying the following conditions:

1. $\mathsf{ExtSetup}$'s output includes parameters, which are distributed identically to $Setup$'s output, and a trapdoor td.
2. The pseudonyms Nym are perfectly binding to Sk and Rn under parameters generated by $\mathsf{ExtSetup}$. Formally, $\forall(Para_{DC}, Sk_{BA}, BL_e, td) \leftarrow \mathsf{ExtSetup}(1^k)$; $\forall Nym$, if $\exists(Aux(Nym), Aux(Nym)')$ satisfying $validAux(Para_{DC}, Nym, Sk, Rn, Aux(Nym)) = true$ and $validAux(Para_{DC}, Nym, Sk', Rn', Aux(Nym)) = true$, then $Sk' = Sk$ and $Rn' = Rn$.
3. From a level L credential proof $CredProof$ honestly produced, $\mathsf{Extract}$ can always extract the corresponding $(F(Sk_O), F(Rn_O), ...F(Sk_L), F(Rn_L)) \leftarrow \mathsf{Extract}(Para_{DC}, td, CredProof, Nym, Nym_O, L)$ of secret keys and r-nyms forming the credential's chain. In the special case for $L = 0$, from any valid Nym, $\mathsf{Extract}$ will output its corresponding $(F(Sk), F(Rn)) \leftarrow \mathsf{Extract}(Para_{DC}, td, \perp, Nym, Nym, 0)$.
4. From a level L credential proof generated by an adversary, $\mathsf{Extract}$ always outputs the correct values for $F(Sk_O), F(Rn_O), F(Sk_L), F(Rn_L)$, or aborts. That means: $Pr[(Para_{DC}, Sk_{BA}, BL_e, td) \leftarrow \mathsf{ExtSetup}(1^k); (CredProof, Nym, Nym_O, L) \leftarrow A(Para_{DC}, td); (f_0, f_0', \ldots, f_L, f_L') \leftarrow \mathsf{Extract}(Para_{DC}, td, CredProof, Nym, Nym_O, L)$: $(f_0, f_0', \ldots, f_L, f_L') \neq \perp \bigwedge ((\exists Sk^*, \exists Rn^*, \exists Aux(Nym)^*$: $validAux(Para_{DC}, Nym, Sk^*, Rn^*, Aux(Nym)^*) = true \wedge (F(Sk^*) \neq f_L \vee F(Rn^*) \neq f_L')) \bigvee (\exists Sk_O^*, \exists Rn_O^*, \exists Aux(Nym_O)^*$: $validAux(Para_{DC}, Nym_O, Sk_O^*, Rn_O^*, Aux(Nym_O)^*) = true \wedge (F(Sk_O^*) \neq f_0 \vee F(Rn_O^*) \neq f_0')))]$ is negligible.
5. The probability that an adversary can forge a valid credential proof from which $\mathsf{Extract}$ returns a chain of identities that is unauthorized or contains a blacklisted identity is negligible. $Pr[(Para_{DC}, Sk_{BA}, BL_e, td) \leftarrow \mathsf{ExtSetup}(1^k); (CredProof, Nym, Nym_O, BL, L) \leftarrow A^{\mathcal{O}(Para_{DC}, command, input)}(Para_{DC}, td); (f_0, f_0', \ldots, f_L, f_L') \leftarrow \mathsf{Extract}(Para_{DC}, td, CredProof, Nym, Nym_O, L) : \mathsf{CredVerify}(Para_{DC}, Nym_O, CredProof, Nym, BL, L) = accept \bigwedge (\exists i$ such that $((f_0, f_0', i, f_{i-1}, f_{i-1}', f_i, f_i') \notin ValidCredentialChains \wedge (f_{i-1}, f_{i-1}') \in HonestUsers) \vee (F^{-1}(f_i') \in BL))]$ is negligible.

where the oracle $\mathcal{O}$ can take the following commands:

- $AddUser$: $\mathcal{O}$ performs $(Sk, Rn) \leftarrow \mathsf{KeyGen}(Para_{DC})$, saves $(Sk, Rn, F(Sk), F(Rn))$ in the user database, stores $(F(Sk), F(Rn))$ in the set $HonestUsers$, and returns $(F(Sk), F(Rn))$ to the adversary.
- $FormNym(f, f')$: $\mathcal{O}$ looks for (Sk, Rn, f, f') in the user database and aborts if it is not there. It generates $(Nym, Aux(Nym)) \leftarrow \mathsf{NymGen}(Para_{DC}, Sk, Rn)$, saves $(Sk, Rn, Nym, Aux(Nym))$ in its pseudonym database, and outputs Nym to the adversary.
- $Issue(Nym_I, Nym_U, Cred_I, DeInf_I, BL, L, Nym_O)$: $\mathcal{O}$ aborts if it cannot find $(Sk_I, Rn_I, Nym_I, Aux(Nym_I))$ or $(Sk_U, Rn_U, Nym_U, Aux(Nym_U))$ in its pseudonym database. Otherwise, it generates $CredProof_I \leftarrow CredProve(Para_{DC}, Nym_O, Cred_I, DeInf_I, Sk_I, Rn_I, Nym_I, Aux(Nym_I), BL, L)$, and extracts $(f_0, f'_0, \ldots, f_L, f'_L) \leftarrow \mathsf{Extract}(Para_{DC}, td, CredProof_I, Nym_I, Nym_O, L)$. $\mathcal{O}$ executes $\mathsf{Issue}(Para_{DC}, Nym_O, Sk_I, Rn_I, Nym_I, Aux(Nym_I), Cred_I, DeInf_I, Nym_U, BL, L) \leftrightarrow \mathsf{Obtain}(Para_{DC}, Nym_O, Sk_U, Rn_U, Nym_U, Aux(Nym_U), Nym_I, BL, L)$ to obtain $(Cred_U, DeInf_U)$. It saves $(f_0, f'_0, L{+}1, f_L, f'_L, F(Sk_U), F(Rn_U))$ in $ValidCredentialChains$ and gives the adversary $(Cred_U, DeInf_U)$.
- $IssueToAdv(Nym_I, Cred_I, DeInf_I, Nym, BL, L, Nym_O)$: $\mathcal{O}$ aborts if it cannot find $(Sk_I, Rn_I, Nym_I, Aux(Nym_I))$ in its pseudonym database. Otherwise, it generates $CredProof_I \leftarrow CredProve(Para_{DC}, Nym_O, Cred_I, DeInf_I, Sk_I, Rn_I, Nym_I, Aux(Nym_I), BL, L)$, and extracts $(f_0, f'_0, \ldots, f_L, f'_L) \leftarrow \mathsf{Extract}(Para_{DC}, td, CredProof_I, Nym_I, Nym_O, L)$ and $(f_{L+1}, f'_{L+1}) \leftarrow \mathsf{Extract}(Para_{DC}, td, \perp, Nym, Nym, 0)$. $\mathcal{O}$ then runs $\mathsf{Issue}(Para_{DC}, Nym_O, Sk_I, Rn_I, Nym_I, Aux(Nym_I), Cred_I, DeInf_I, Nym, BL, L)$ to interact with the adversary and saves $(f_0, f'_0, L+1, f_L, f'_L, f_{L+1}, f'_{L+1})$ in $ValidCredentialChains$ if the protocol ends successfully.
- $ObtainFromAdv(Nym, Nym_U, Nym_O, BL, L)$: $\mathcal{O}$ aborts if it cannot find $(Sk_U, Rn_U, Nym_U, Aux(Nym_U))$ in its pseudonym database. Otherwise, it executes $\mathsf{Obtain}(Para_{DC}, Nym_O, Sk_U, Rn_U, Nym_U, Aux(Nym_U), Nym, BL, L)$ to interact with the adversary to return $(Cred_U, DeInf_U)$.
- $Prove(Nym, Cred, DeInf, Nym_O, BL, L)$: $\mathcal{O}$ aborts if it cannot find $(Sk, Rn, Nym, Aux(Nym))$ in its pseudonym database. Otherwise, it executes $CredProve(Para_{DC}, Nym_O, Cred, DeInf, Sk, Rn, Nym, Aux(Nym), BL, L)$ and returns the output.
- $Revoke(Sk, Rn, BL)$: The oracle aborts if it cannot find $(F(Sk), F(Rn))$ in the set $HonestUsers$. Otherwise, it recomputes BL to blacklist Rn.

10.4 Delegability and Anonymity

Delegability and anonymity do not always go together, such as in this case. Suppose user I delegates to user U the ability to prove that I is not revoked in BL, and U knows I by Nym_I. Then, in any construction, given an r-nym Rn, U and BA can collude to tell if Rn belongs to Nym_I or not by blacklisting Rn and checking if U can still prove that I is not revoked. So it is important that a user keeps her r-nym *secret* and she should know that such delegation could compromise her anonymity when issuing, as explained. It is still her right to or not to delegate that proving ability (by issuing $DeInf$ or not).

Even then, we emphasize that in worst cases, the only privacy lost is that a collision of BA and the delegatee could learn if an r-nym belongs to a delegator from $\mathsf{Issue} \leftrightarrow \mathsf{Obtain}$. Other privacy properties, such as anonymity of $CredProof$, Nym and the delegatee, are still maintained.

This limitation is related to the restriction on ADNMP mentioned in section 4. When a BL is implemented by using ADNMP to accumulate revoked Rns, given an Rn' and an ADNMP delegating key De, a user can collude with BA to tell if De is generated by Rn'. This limitation is also reflected in the Anonymity definition in Appendix 10. For the case that $DeInf$ is included, when interacting with SimIssue, r-nyms on the chain of issuer's credentials are randomly generated and not revealed to the adversary, because as discussed above, a user and BA can tell if a given r-nym belongs to one of the delegators on her chain.

11 Extra Details on the RDAC System

11.1 Instantiation of Building Blocks

We could use the SXDH instantiation of a *secure* ADNMP presented in section 5, though the SDLIN instantiation is also possible. AcSetup generates $Para_{Ac} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, P_1, P_2, A, \sigma, \varsigma, \tau)$ and $Sk_{BA} = Aux_{Ac} = \delta \leftarrow \mathbb{Z}_p^*$. The accumulator domain is $\mathbb{D} = \mathbb{Z}_p^* \backslash \{-\delta\}$.

Its NMPS (AcSetup, ProveNM, VerifyNM) is a GS proof system, which is *randomizable, composable ZK* and delegatable using (Dele, Rede, Vali, CompNMProof). ComNM is the normal GS commitment ComGS in the SXDH instantiation. So a delegating key De contains a commitment of its element Ele. And CompNMProof and Rede generate Ele's commitment in their outputs by randomizing the commitment in De.

Regarding BCCKLS's CredPS (PKSetup, PKProve, PKVerify, RandProof), PKSetup's output is (p, $\mathbb{G}_1$, $\mathbb{G}_2$, $\mathbb{G}_T$, e, P_1, P_2, σ), which is generated by AcSetup. This is a GS proof system in the SXDH instantiation, *randomizable, composable ZK* and *partially extractable* for $(F(Sk_O), F(Rn_0), F(Sk_1), F(Rn_1), ..., F(Sk_L), F(Rn_L))$ of a credential proof level L. A *collision resistant* H, an *F-unforgeable and certification-secure* $\mathcal{AU}$ and its secure AuthPro could be instantiated the same as in [1]. They all share the same bilinear pairing parameters, so elements Rn of the accumulator domain and the authenticator's keys Sk are in $\mathbb{Z}_p$ and committable by Com.

An EQPS instantiation can be constructed based on [12, 1]. In [12], BCKL propose an $NIZK_{GS}$ to prove that two given GS commitments are committed to the same value. In BCCKLS's CredPS, commitment $\mathsf{Com}(y)$ of a variable y consists of two GS commitments $comGS_{yA} \leftarrow \mathsf{ComGS}(yA)$ of yA and $\mathsf{ComGS}(yB)$ of yB, where public parameters $A \in \mathbb{G}_1$ and $B \in \mathbb{G}_2$, and an $NIZK_{GS}$ that these are commitments to the same value y. EQPS proves that y is also committed in an NMPS commitment $comNM_y \leftarrow \mathsf{ComNM}(y)$, which is also $\mathsf{ComGS}(y)$, as follows. It generates and concatenates: a GS proof of equation $X - yA = 0$ using new GS commitments $comGS_y$ of y and $comGS_X$ of X; a BCKL $NIZK_{GS}$ that $comGS_{yA}$ and $comGS_X$ commit to the same value; and a BCKL $NIZK_{GS}$ that $comNM_y$ and $comGS_y$ commit to the same value.

EQPS is *randomizable* and *composable ZK*. Given commitments $\mathsf{Com}(y)$ and $\mathsf{ComNM}(y)$ of y, the simulator picks a random y' and computes $X' = y'A$. It generates and outputs a concatenation of: a GS proof of equation $X' - y'A = 0$ using new GS commitments $comGS_{y'}$ of y' and $comGS_{X'}$ of X'; a simulation of BCKL $NIZK_{GS}$ that $comGS_{yA}$ of $\mathsf{Com}(y)$ and $comGS_{X'}$ commit to the same value; and a simulation of BCKL $NIZK_{GS}$ that $\mathsf{ComNM}(y)$ and $comGS_{y'}$ commit to the same value. The simulation is indistinguishable from the a real proof due to the BCKL $NIZK_{GS}$ is composable ZK and ComGS is hiding.

11.2 Exposing R-nyms

We can use the following methods for BA to obtain r-nyms to revoke.

– There is a Issuing Authority (IA) who issues user r-nyms and makes requests to BA to revoke r-nyms. An r-nym Rn is not generated for an user by KeyGen. Instead, there is a protocol
  $\mathsf{IssueRnym}(Para_{DC}, Sk_{IA}) \leftrightarrow \mathsf{ObtainRnym}(Para_{DC}, Sk)$ between IA with a secret key Sk_{IA} and the user with a secret key Sk, by which the user obtains a revocation nym Rn and a proof that Rn and Sk are authenticated by IA and IA gets and stores Rn in a database DB_{Rn}. Rn is then a secret known only to the user and IA. IA can send some r-nyms to BA to revoke. CredProve also needs to prove that R-nyms and secret keys of all users on the chain are authenticated by IA. IA just needs to be trusted only for Anonymity, not for Unforgeability and Correctness.
  An IA can be added to the RDAC construction as follows. Setup also generates a secret key Sk_{IA} of the authentication scheme $\mathcal{AU}$ and a public key $Pk_{IA} \leftarrow \mathsf{Com}(Sk_{IA})$ for IA.
  In the protocol IssueRnym $\leftrightarrow$ ObtainRnym, IA generates a new r-nym Rn and run AuthPro for the user with secret key Sk to get Rn and an NIZKPK that Rn and Sk are authenticated by IA
  $RnProof \leftarrow NIZKPK[Sk_{IA} \text{ in } Pk_{IA}, Sk \text{ in } \mathsf{Com}(Sk, 0), Rn \text{ in } \mathsf{Com}(Rn, 0)]$
  $\{(F(Sk_{IA}), F(Sk), F(Rn), auth) : \mathsf{VerifyAuth}(Sk_{IA}, (Sk, Rn), auth)\}$. CredProve needs to concatenate randomizations of these proofs to extend its output proof (1) as
  $CredProof \leftarrow \mathsf{NIZKPK}[Sk_O \text{ in } Nym_O[1], Sk \text{ in } Nym[1], Rn \text{ in } Nym[2], Sk_{IA}$ in $Pk_{IA}]$
  $\{(F(Sk_O), F(Sk_1), F(Rn_1), ..., F(Sk_L), F(Rn_L), F(Sk), F(Rn), F(Sk_{IA}),$
  $auth_1, ..., auth_L, auth_{L+1}, auth'_1, ..., auth'_L, auth'_{L+1}) :$
  $\mathsf{VerifyAuth}(Sk_O, (Sk_1, Rn_1, r_1), auth_1) \wedge (Rn_1 \notin BL) \wedge \mathsf{VerifyAuth}(Sk_{IA}, (Sk_1,$
  $Rn_1), auth'_1) \wedge$
  $\mathsf{VerifyAuth}(Sk_1, (Sk_2, Rn_2, r_2), auth_2) \wedge (Rn_2 \notin BL) \wedge \mathsf{VerifyAuth}(Sk_{IA}, (Sk_2,$

$Rn_2), auth'_2) \wedge ... \wedge$
$\mathsf{VerifyAuth}(Sk_{L-1}, (Sk_L, Rn_L, r_L), auth_L) \wedge (Rn_L \notin BL) \wedge \mathsf{VerifyAuth}(Sk_{IA},$
$(Sk_L,\ Rn_L),\ auth'_L) \wedge$
$\mathsf{VerifyAuth}(Sk_L, (Sk, Rn, r_{L+1}), auth_{L+1}) \wedge (Rn \notin BL) \wedge \mathsf{VerifyAuth}(Sk_{IA},$
$(Sk,\ Rn),\ auth'_{L+1})\}$.

- This is a method adopted from group signatures [?]. There is a Managing Authority (MA), who can *open* any credential proof of a misbehaved or disputed prover to find its r-nym for BA to revoke, and also plays as the IA above. There is another algorithm
  $\mathsf{OpenRnym}(Para_{DC}, Sk_{MA}, DB_{Rn}, CredProof)$ that takes MA's secret key Sk_{MA}, r-nym database DB_{Rn} and a credential proof $CredProof$ and outputs the credential prover's r-nym. MA just needs to be trusted only for Anonymity, not for Unforgeability and Correctness.
  In the RDAC instantiation above, the IA could be extended to be an MA by using the Linear Encryption (LE) scheme [?], which is semantically secure against a chosen plaintext attack based on DLIN, as follow. It extends the public bilinear pairing parameters $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, P_1, P_2)$ with the LE public key consisting of $U, V, H, G \leftarrow \mathbb{G}_1$ such that there are $k, l \leftarrow \mathbb{Z}_p$ satisfying $kU = lV = H$. MA's private key now also includes k, l.
  Each time generating a credential proof, the prover with r-nym y also returns an LE encryption $(T_1 = rU, T_2 = sV, T_3 = yG + (r + s)H)$, where $r, s \leftarrow \mathbb{Z}_p$, and concatenates a GS proof of these equations with variables r, s, y to its output proof.
  When MA needs to open a credential proof, it decrypts the LE ciphertext to get $yG = T_3 - kT_1 - lT_2$, which is used to look up DB_{Rn} to find r-nym y.
- There could be an authority who could force any user to reveal his r-nym to BA and prove his ownership of the r-nym by using CredProve and showing openings of his r-nym's commitment. For example, users may be required to give deposits to the authority when entering the system. If an user does not follow the enforcement, he loses his deposit.

12 Security Proofs for the RDAC System

This section presents security proofs for the proposed Revocable Delegatable Anonymous Credential system, based on security proofs for the BCCKLS delegatable anonymous credential scheme [1]. The proofs are provided for the more general case, where a user must prove that her credential and all of its ancestors are not revoked. The other case, where a user only has to prove that her credential is not revoked, is simpler and can be derived from this.

12.1 Correctness

The scheme meets the Correctness requirements as follows.

1. This requirement holds based on the definitions of *validAux* and NymGen.

2. The verifications by the issuer on Nym_I, $Cred$, $DeInf$, Sk_I, Rn_I and Nym_U at the beginning of Issue $\leftrightarrow$ Obtain make sure that Issue aborts if one of the predicates $properPair$ and $validAux$s fails.
3. Obtain terminates unsuccessfully if Nym_U is not correctly computed from Sk_U, Rn_U and
$Aux(Nym_U)$, or if AuthPro fails, or if PKVerify on $CredProof_I$ rejects, or if one of r-nyms on the chain is blacklisted, or if verifying $DeInf_U$, which verifies the validity of its delegating keys by using Vali and the correctness of the relationship between $Cred_U$ and $DeInf_U$ by checking $EQProof_j$, fails. And $Cred_U$ is formed from AuthPro's output and $CredProof_I$. So if Obtain ends successfully, its output is a proper pair.
4. As indicated above, either Issue or Obtain aborts if their inputs are invalid. If their inputs are 'proper' and they execute honestly, Obtain outputs a proper pair, based on Completeness of CredPS, NMPS, EQPS and $\mathcal{AU}$, and Delegability and Redelegability of the accumulator.
5. CredProve first verifies validity of Nym, $Cred$ and $DeInf$ with regard to Sk, Rn, $Aux(Nym)$ and Nym_O. It also cannot generate a proof if Rn or one of Rn_j is blacklisted, due to Delegability and Redelegability of the accumulator. So if $Cred$ and $DeInf$ is not 'proper' or Nym is invalid, CredProve will abort. Otherwise, it should generate a proof accepted by CredVerify, as in the definition of $properPair$.

12.2 Anonymity

We prove that the proposed RDAC scheme provides Anonymity if the building blocks are secure. The simulator algorithms are defined as follows:

- SimSetup(1^k): As CredPS, NMPS and EQPS are composable ZK and EQPS's setup consists of PKSetup of CredPS and AcSetup of NMPS, there is a simulation setup algorithm SimConSetup for the combination of PKSetup and AcSetup. SimSetup first uses AtSetup(1^k) to generate $Para_{At}$ for an F-unforgeable certificaton secure authentication scheme; then uses SimConSetup to generate corresponding $Para_{PK}$, $Para_{Ac}$, Aux_{Ac} and trapdoor sim. Let H be a collision resistant hash function whose output range is the authentication scheme's message space. The output includes an empty BL_e, $Para_{DC} = (Para_{PK}, Para_{At}, Para_{Ac}, H)$, $Sk_{BA} = Aux_{Ac}$ and sim.
- SimProve($Para_{DC}, sim, Nym_O, Nym, BL, L, flag$): Abort if $flag = false$. Otherwise, generate random Sk_1,...,Sk_{L-1} and Rn_1,...,$Rn_{L-1} \notin BL$ and their nyms Nym_1,...,Nym_{L-1}; let $Nym_0 = Nym_O$ and $Nym_L = Nym$; and get $r_1, ..., r_L$ where $r_i = H(Nym_O, attributes, i)$. Run the NIZKPK simulators with the trapdoor sim for the composable ZK CredPS, NMPS and EQPS to output simulated proofs π_1,...,π_L as follows
$\pi_i \leftarrow SimNIZKPK[Sk_{i-1}$ in $Nym_{i-1}[1], Sk_i$ in $Nym_i[1], Rn_i$ in $Nym_i[2]]$
$\{(F(Sk_{i-1}), F(Sk_i), F(Rn_i), auth_i)$:
VerifyAuth$(Sk_{i-1}, (Sk_i, Rn_i, r_i), auth_i) \wedge (Rn_i \notin BL)\}$
Concatenate $\pi_1 \circ ... \circ \pi_L$ and return $Nym_0, ..., Nym_L$ with the concatenation.

- $\mathsf{SimObtain}(Para_{DC}, sim, Nym_O, Nym, Nym_A, BL, L, flag)$: Abort if $flag = false$.
  Otherwise, given $CredProof_I$, $NMChainProof$ and the proof that $Rn_A \notin BL$ from the adversary, verify that $\mathsf{PKVerify}(Para_{PK}, (Nym_0, Nym_A), CredProof_I)$ accepts and the other two proofs pass. Compute $r_L = H(Nym_O, atributes, L)$ and simulate the two party computation protocol $\mathsf{AuthPro}$ with the adversary to obtain a proof of knowledge of an authentication tag for (Sk, Rn, r_L). Compute $Cred_U$ as in Obtain and also receive $DeInf$ which is totally generated by A_2.
- $\mathsf{SimIssue}(Para_{DC}, sim, Nym_O, Nym, Nym_A, BL, L, flag)$: Abort if $flag = false$. Otherwise, generate random r-nyms Rn_1,...,$Rn_L \notin BL$ and random Sk_1,...,Sk_{L-1}; compute their L delegating keys $De_i \leftarrow \mathsf{Dele}(Para_{Ac}, Rn_i)$ and their nyms Nym_1,...,Nym_{L-1}; let $Nym_0 = Nym_O$, $Nym_L = Nym$ and $Nym_{L+1} = Nym_A$; and get $r_1, ..., r_{L+1}$ where $r_i = H(Nym_O, attributes, i)$. Run the NIZKPK simulators with the trapdoor sim for the composable ZK CredPS and EQPS to simulate proofs $Proof_1$,...,$Proof_{L+1}$, $EQProof_1$, ..., $EQProof_L$ as follows
  $Proof_i \leftarrow SimNIZKPK_{CredPS}[Sk_{i-1}$ in $Nym_{i-1}[1], Sk_i$ in $Nym_i[1], Rn_i$ in $Nym_i[2]]$
  $\{(F(Sk_{i-1}), F(Sk_i), F(Rn_i), auth_i) :$
  $\mathsf{VerifyAuth}(Sk_{i-1}, (Sk_i, Rn_i, r_i), auth_i)\}$ and
  $EQProof_i \leftarrow SimNIZKPK_{EQPS}[Rn_i$ in $Nym_i[2], Rn'_i$ in $De_i]$
  $\{(F(Rn_i), F(Rn'_i)) : Rn_i = Rn'_i\}$
  It sends the adversary $Nym_0, ..., Nym_L$, and $CredProof_I = Proof_1 \circ ... \circ Proof_L$, and $DeInf_U$ including De_1,...,De_L and $EQProof_1 \circ ... \circ EQProof_L$, and simulates the two party computation protocol $\mathsf{AuthPro}$ to give $Proof_{L+1}$ to the adversary.

We see that the accumulator's 4 delegation properties still hold under parameters generated by $\mathsf{SimSetup}$, otherwise an adversary breaking one of the properties could distinguish $\mathsf{SimSetup}$ and Setup. The followings show that these algorithms satisfy the required properties, using similar arguments from [1].

1. It holds as CredPS, NMPS and EQPS are composable ZK.
2. It holds as the commitment schemes are hiding.
3. Under the strong computational hiding property of the commitment schemes, the distribution of the Nym commitments generated by $\mathsf{SimProve}$ is indistinguishable from the honest Nym commitments generated by $\mathsf{CredProve}$. Due to Delegability and Redelegability of the ADNMP scheme, the NMPS proofs generated using $\mathsf{CompNMProof}$ with $DeInf$ have the same distribution as those generated using $\mathsf{ProveNM}$ with r-nyms. Additionally, the proof systems are composable ZK, so the simulated proofs generated by $SimNIZKPK$ are indistinguishable from those generated by $\mathsf{CredProve}$.
4. The difference between behaviors of $\mathsf{SimObtain}$ and Obtain is that $\mathsf{SimObtain}$ performs the $\mathsf{AuthPro}$ protocol using its simulation. That is indistinguishable from an honest Obtain's $\mathsf{AuthPro}$ performance, due to $\mathsf{AuthPro}$'s security.

5. The 5 differences between SimIssue and Issue are as follows. First, SimIssue's delegating keys De_i are generated by r-nyms, which could be different from those to generate Issue's delegating keys. Second, SimIssue's $EQProof_i$ are generated by $SimNIZKPK_{EQPS}$ instead of, as in Issue, rerandomizing the EQ proofs in the issuer's $DeInf$ or generating $EQProof_L$. Third, SimIssue's $CredProof_I$ is formed using $SimNIZKPK_{CredPS}$ instead of by rerandomizing a real $Cred$. Next, SimIssue simulates the AuthPro protocol. Finally, SimIssue's $Proof_{L+1}$ is generated by $SimNIZKPK_{CredPS}$ whereas Issue uses the AuthPro protocol. SimIssue and Issue are indistinguishable, despite of those differences, due to the following reasons:
   – Based on the accumulator's Unlinkability, Delegability and Redelegability, the adversary can not distinguish the delegating key lists generated by SimIssue and Issue, as r-nyms of input $DeInf$ are randomly generated and not revealed to the adversary.
   – Outputs of $NIZKPK_{EQPS}$ and $SimNIZKPK_{EQPS}$ are indistinguishable.
   – Outputs of $NIZKPK_{CredPS}$ and $SimNIZKPK_{CredPS}$ are indistinguishable.
   – The AuthPro protocol is secured, so its real and simulated executions are indistinguishable.

12.3 Unforgeability

The unforgeability proof is similar to the one in [1], based on F-unforgeability and certification-security of the authentication scheme, partial extractability of CredPS, and soundness of NMPS and EQPS. ExtSetup is constructed identically to Setup except that the extraction setup of the partially extractable CredPS is used to generate $Para_{PK}$, $Para_{Ac}$ and td. Extract is CredPS's witness extractor. We show that the requirements are satisfied as follows.

1. It holds as the output of CredPS's extraction setup, without td, is indistinguishable from the output of its real setup.
2. A pseudonym is computed as the commitment of Sk and Rn using a perfectly binding commitment scheme, so the pseudonyms are perfectly binding to Sk and Rn.
3. CredPS is partly a proof of knowledge of $(F(Sk_O), F(Rn_O), ...F(Sk_L), F(Rn_L))$ for a level L credential proof. So Extract can extract these values from an honest credential proof. In case $L = 0$, it can extract $(F(Sk_O), F(Rn_O))$ from its valid commitment Nym_O.
4. Similarly, if an adversary generates a $CredProof$ accepted by CredVerify, then Extract can extract $(F(Sk_O), F(Rn_O), F(Sk_L), F(Rn_L))$. Otherwise, it aborts.
5. Suppose an adversary can win the real game G defined in this requirement, that means it can forge a credential proof level L accepted by CredVerify such that either of the following cases happens.

- Case 1: The probability that there exists $i \leq L$ satisfying $(f_0, f'_0, i, f_{i-1}, f'_{i-1}, f_i, f'_i) \notin ValidCredentialChains \wedge (f_{i-1}, f'_{i-1}) \in HonestUsers$ is non negligible.
- Case 2: The probability that there exists $i \leq L$ satisfying $F^{-1}(f'_i) \in BL$ is non negligible.

We show that in Case 1, the adversary can be used to break the authentication scheme's security, and in Case 2, the adversary can be used to break soundness of NMPS or EQPS.

Case1: Similar to [1], we let the adversary play the following game G', which is indistinguishable from the real game G. We then show that if an adversary can win this game, it can break the authentication scheme's security. In G', choose a random user u who is given (Sk^*, Rn^*) by the $AddUser$ query. G' then proceeds the same as G, except that for queries $IssueToAdv$ and $ObtainFromAdv$ on user u, the simulator of the AuthPro protocol is used instead of the real one. That means when receiving $IssueToAdv(Nym, Cred, DeInf, Nym_A, BL, L, Nym_O)$ or $ObtainFromAdv(Nym_A, Nym, Nym_O, BL, L)$, if $\mathcal{O}$ can find $(Sk, Rn, Nym, Aux(Nym))$ in its pseudonym database and $(Sk, Rn) = (Sk^*, Rn^*)$, it uses AuthPro's simulator to simulate the interaction. Then due to AuthPro's security, G' is computationally indistinguishable from G. Now assume that a PPT adversary A can win in G', we show that A can be used to build a PPT adversary B which can break the authentication scheme $\mathcal{AU}$'s security. Suppose B is given $\mathcal{AU}$'s challenge $Para_{At}$, $f^* = F(Sk^*)$ and oracles $\mathcal{O}_{Authen}(Para_{At}, Sk^*, \cdot)$ and $\mathcal{O}_{Certify}(Para_{At}, \cdot, (Sk^*, \cdots))$ where the secret key Sk^* is not known to B. B uses $Para_{At}$ as part of and creates other parameters in $Para_{DC}$ and a trapdoor td. B gives $Para_{DC}$ and td to A and responds to A's oracle queries as follows.

(a) $AddUser$: B randomly chooses a query where its answer to A is $(F(Sk^*), F(Rn^*))$ for some Rn^* and it saves $(\square, Rn^*, F(Sk^*), F(Rn^*))$ in the user database and $F(Sk^*), F(Rn^*))$ in the $HonestUsers$ set ('$\square$' indicates the unknown challenge key Sk^*). For other queries, it runs as a normal $AddUser$.

(b) $FormNym(f, f')$: If $f \neq f^*$, B runs as the defined $FormNym$. Otherwise it generates a random $Aux(Nym)$ to compute Nym, saves $(\square, Rn^*, Nym, Aux(Nym))$ in its pseudonym database and returns Nym to A.

(c) $Issue(Nym_I, Nym_U, Cred_I, DeInf_I, BL, L, Nym_O)$: B aborts if it cannot find $(Sk_U, Rn_U, Nym_U, Aux(Nym_U))$ and $(Sk_I, Rn_I, Nym_I, Aux(Nym_I))$ in pseudonym database or $Sk_U = Sk_I$. Otherwise, if $F(Sk_U) \neq f^*$ and $F(Sk_I) \neq f^*$, B runs as the defined oracle. In the two cases $F(Sk_I) = f^*$ or $F(Sk_U) = f^*$, it executes as defined except that it uses $\mathcal{O}_{Authen}(Para_{At}, Sk^*, (Sk_U, Rn_U, H(Nym_O, attributes, L)))$ or $\mathcal{O}_{Certify}(Para_{At}, Sk_I, (Sk^*, Rn^*, H(Nym_O, attributes, L)))$, respectively, to compute the credential proof.

(d) $IssueToAdv(Nym_I, Cred_I, DeInf_I, Nym, BL, L, Nym_O)$: B aborts if it cannot find $(Sk_I, Rn_I, Nym_I, Aux(Nym_I))$ in its pseudonym database. Otherwise, if $F(Sk_I) \neq f^*$, B runs as the defined oracle. In the case

$F(Sk_I) = f^*$, it executes as defined except that it simulates the AuthPro protocol and uses $\mathcal{O}_{Authen}$.

(e) $ObtainFromAdv(Nym, Nym_U, Nym_O, BL, L)$: B aborts if it cannot find $(Sk_U, Rn_U, Nym_U, Aux(Nym_U))$ in its pseudonym database. Otherwise, if $F(Sk_U) \neq f^*$, B runs as the defined oracle. In the case $F(Sk_U) = f^*$, it executes as defined except that it simulates the AuthPro protocol and uses $\mathcal{O}_{Certify}$.

(f) $Prove(Nym, Cred, DeInf, Nym_O, BL, L)$: B does not need Sk^* for this query, so it can execute and output as the defined oracle.

(g) $Revoke(Sk, Rn, BL)$: Again, B does not need Sk^* for this query, so it can execute and output as the defined oracle.

So B can respond to all queries from A. Thus, A can output with non-negligibility $(CredProof, Nym, Nym_O, BL, L)$ such that CredVerify$(Para_{DC}$, Nym_O, $CredProof$, Nym, BL, $L) = accept$ and $\exists i$ such that $(f_0, f'_0, i, f_{i-1}, f'_{i-1}, f_i, f'_i) \notin ValidCredentialChains \wedge (f_{i-1}, f'_{i-1}) \in HonestUsers$, where $(f_0, f'_0, \ldots, f_L, f'_L) \leftarrow$ Extract$(Para_{DC}, td, CredProof, Nym, Nym_O, L)$. As the number of $AddUser$'s queries is bound, and A has no knowledge about B's randomly chosen $AddUser$ query for injecting $F(Sk^*)$, the probability $f_{i-1} = f^*$ is non-negligible. In such case, B can extract $auth_i$ such that VerifyAuth$(Sk^*$, $(Sk_i$, Rn_i, $H(Nym_O, attributes, i)), auth_i)$ accepts. That forgery shows that B has broken $\mathcal{AU}$'s security.

Case2: Similarly, we show that from an adversary A of this case, we can construct an adversary B to break the soundness of NMPS or EQPS. B can perfectly simulate answers to A's oracle queries $AddUser$, $FormNym$, $Issue$, $IssueToAdv$, $ObtainFromAdv$, $Prove$ and $Revoke$, as the only secret B does not know is $Sk_{BA} = Aux_{Ac}$, which is not required to answer any of these queries.

References

1. Belenkiy, M., Camenisch, J., Chase, M., Kohlweiss, M., Lysyanskaya, A., Shacham, H.: Randomizable proofs and delegatable anonymous credentials. In Halevi, S., ed.: CRYPTO 2009. Volume 5677 of LNCS., Santa Barbara, CA, USA, Springer, Berlin, Germany (August 16–20, 2009) 108–125
2. Groth, J., Sahai, A.: Efficient non-interactive proof systems for bilinear groups. In Smart, N.P., ed.: EUROCRYPT 2008. Volume 4965 of LNCS., Istanbul, Turkey, Springer, Berlin, Germany (April 13–17, 2008) 415–432
3. Bellare, M., Rogaway, P.: Random oracles are practical: A paradigm for designing efficient protocols. In Ashby, V., ed.: ACM CCS 93, Fairfax, Virginia, USA, ACM Press (November 3–5, 1993) 62–73
4. Benaloh, J.C., de Mare, M.: One-way accumulators: A decentralized alternative to digital sinatures (extended abstract). In Helleseth, T., ed.: EUROCRYPT'93. Volume 765 of LNCS., Lofthus, Norway, Springer, Berlin, Germany (May 23–27, 1993) 274–285
5. Bari, N., Pfitzmann, B.: Collision-free accumulators and fail-stop signature schemes without trees. In Fumy, W., ed.: EUROCRYPT'97. Volume 1233 of LNCS., Konstanz, Germany, Springer, Berlin, Germany (May 11–15, 1997) 480–494

6. Camenisch, J., Lysyanskaya, A.: Dynamic accumulators and application to efficient revocation of anonymous credentials. In Yung, M., ed.: CRYPTO 2002. Volume 2442 of LNCS., Santa Barbara, CA, USA, Springer, Berlin, Germany (August 18–22, 2002) 61–76
7. Nguyen, L.: Accumulators from bilinear pairings and applications. In Menezes, A., ed.: CT-RSA 2005. Volume 3376 of LNCS., San Francisco, CA, USA, Springer, Berlin, Germany (February 14–18, 2005) 275–292
8. Camenisch, J., Kohlweiss, M., Soriente, C.: An accumulator based on bilinear maps and efficient revocation for anonymous credentials. In Jarecki, S., Tsudik, G., eds.: PKC 2009. Volume 5443 of LNCS., Irvine, CA, USA, Springer, Berlin, Germany (March 18–20, 2009) 481–500
9. Li, J., Li, N., Xue, R.: Universal accumulators with efficient nonmembership proofs. In Katz, J., Yung, M., eds.: ACNS 07. Volume 4521 of LNCS., Zhuhai, China, Springer, Berlin, Germany (June 5–8, 2007) 253–269
10. Au, M.H., Tsang, P.P., Susilo, W., Mu, Y.: Dynamic universal accumulators for DDH groups and their application to attribute-based anonymous credential systems. In Fischlin, M., ed.: CT-RSA 2009. Volume 5473 of LNCS., San Francisco, CA, USA, Springer, Berlin, Germany (April 20–24, 2009) 295–308
11. Camenisch, J., Lysyanskaya, A.: Signature schemes and anonymous credentials from bilinear maps. In Franklin, M., ed.: CRYPTO 2004. Volume 3152 of LNCS., Santa Barbara, CA, USA, Springer, Berlin, Germany (August 15–19, 2004) 56–72
12. Belenkiy, M., Chase, M., Kohlweiss, M., Lysyanskaya, A.: P-signatures and non-interactive anonymous credentials. In Canetti, R., ed.: TCC 2008. Volume 4948 of LNCS., San Francisco, CA, USA, Springer, Berlin, Germany (March 19–21, 2008) 356–374
13. Brickell, E.F., Camenisch, J., Chen, L.: Direct anonymous attestation. In Atluri, V., Pfitzmann, B., McDaniel, P., eds.: ACM CCS 04, Washington D.C., USA, ACM Press (October 25–29, 2004) 132–145
14. Camenisch, J., Van Herreweghen, E.: Design and implementation of the idemix anonymous credential system. In Atluri, V., ed.: ACM CCS 02, Washington D.C., USA, ACM Press (November 18–22, 2002) 21–30
15. Microsoft: U-prove community technology preview. In: https://connect.microsoft.com/. (2010)
16. Bichsel, P., Camenisch, J., Groß, T., Shoup, V.: Anonymous credentials on a standard java card. In Al-Shaer, E., Jha, S., Keromytis, A.D., eds.: ACM CCS 09, Chicago, Illinois, USA, ACM Press (November 9–13, 2009) 600–610
17. Tsang, P.P., Au, M.H., Kapadia, A., Smith, S.W.: PEREA: towards practical TTP-free revocation in anonymous authentication. In Ning, P., Syverson, P.F., Jha, S., eds.: ACM CCS 08, Alexandria, Virginia, USA, ACM Press (October 27–31, 2008) 333–344
18. Johnson, R., Molnar, D., Song, D.X., Wagner, D.: Homomorphic signature schemes. In Preneel, B., ed.: CT-RSA 2002. Volume 2271 of LNCS., San Jose, CA, USA, Springer, Berlin, Germany (February 18–22, 2002) 244–262
19. Ateniese, G., Kamara, S., Katz, J.: Proofs of storage from homomorphic identification protocols. In Matsui, M., ed.: ASIACRYPT 2009. Volume 5912 of LNCS., Tokyo, Japan, Springer, Berlin, Germany (December 6–10, 2009) 319–333
20. Charles, D., Jain, K., Lauter, K.: Signatures for network coding. In: International Journal on Information and Coding Theory. (2006)
21. Yun, A., Cheon, J., Kim, Y.: On homomorphic signatures for network coding. In: Transactions on Computer. (2009)

22. Johnson, R., Walsh, L., Lamb, M.: Homomorphic signatures for digital photographs. In: Suny Stony Brook. (2008)
23. Monnerat, J., Vaudenay, S.: Generic homomorphic undeniable signatures. In Lee, P.J., ed.: ASIACRYPT 2004. Volume 3329 of LNCS., Jeju Island, Korea, Springer, Berlin, Germany (December 5–9, 2004) 354–371
24. Groth, J.: A verifiable secret shuffle of homomorphic encryptions. In Desmedt, Y., ed.: PKC 2003. Volume 2567 of LNCS., Miami, USA, Springer, Berlin, Germany (January 6–8, 2003) 145–160
25. Fouque, P.A., Poupard, G., Stern, J.: Sharing decryption in the context of voting or lotteries. In Frankel, Y., ed.: FC 2000. Volume 1962 of LNCS., Anguilla, British West Indies, Springer, Berlin, Germany (February 20–24, 2000) 90–104
26. Gentry, C.: Fully homomorphic encryption using ideal lattices. In Mitzenmacher, M., ed.: 41st ACM STOC, Bethesda, Maryland, USA, ACM Press (May 17–20, 2009) 169–178
27. Dodis, Y., Haralambiev, K., Lopez-Alt, A., Wichs, D.: Cryptography against continuous memory attacks (2010)
28. Ghadafi, E., Smart, N., Warinschi, B.: Groth sahai proofs revisited. In: PKC. (2010)
29. Acar, T., Nguyen, L.: Revocation for delegatable anonymous credentials. Technical Report MSR-TR-2010-170, Microsoft Research, One Microsoft Way, Redmond, WA 98052 (December 2010)
30. Boyen, X., Waters, B.: Full-domain subgroup hiding and constant-size group signatures. In Okamoto, T., Wang, X., eds.: PKC 2007. Volume 4450 of LNCS., Beijing, China, Springer, Berlin, Germany (April 16–20, 2007) 1–15
31. Boyen, X.: The uber-assumption family (invited talk). In Galbraith, S.D., Paterson, K.G., eds.: PAIRING 2008. Volume 5209 of LNCS., Egham, UK, Springer, Berlin, Germany (September 1–3, 2008) 39–56
32. Camenisch, J., Neven, G., Shelat, A.: Simulatable adaptive oblivious transfer. In Naor, M., ed.: EUROCRYPT 2007. Volume 4515 of LNCS., Barcelona, Spain, Springer, Berlin, Germany (May 20–24, 2007) 573–590