Homomorphic Proofs and Applications

Tolga Acar
tolga@microsoft.com

Lan Nguyen
languyen@microsoft.com

Microsoft Research
One Microsoft Way, Redmond, WA 98052, USA

Categories and Subject Descriptors
E.3 [**Data Encryption**]: Public Key Crypto Systems

General Terms
Security, Cryptography, Theory

Keywords
Privacy, Anonymous Credential, Revocation, Accumulator, Proof System, Zero Knowledge.

ABSTRACT
This paper introduces and formalizes *homomorphic proofs*, which allow 'adding' proofs and 'adding' their statements to get a new proof of the 'sum' statement. We propose a construction of homomorphic proofs and show one of its applications which is a new *accumulator scheme with delegatable non-membership (NM) proofs*. We use this accumulator in extending the BCCKLS scheme [3] to achieve a *delegatable anonymous credential with revocation* system. Informally, the accumulator's delegatable 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 which could even be updated afterward. Moreover, the delegation should be redelegatable, unlinkable and verifiable. Security of the proposed schemes is provable. We implement a system for Revocation of Anonymous Credentials (*RAC*) that bases on the proposed accumulator. RAC could provide revocation for several anonymous credential systems.

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 [18] 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. They can be randomized, i.e. generating a new proof from an existing proof of the same statement without knowing the witness. In this paper, we will unveil another valuable feature of GS proofs, homomorphism.

Proof systems are used to construct *accumulators*. An accumulator allows aggregation of a large set of elements into one constant-size *accumulator value*. There is a proof system, called 'membership', to prove that an element is accumulated. An accumulator is said to be *universal* if there is another proof system, called 'non-membership', to prove that a given element is not accumulated in the accumulator value. An accumulator is said to be 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 [2]. Some 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*.

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 [10, 4, 3]; applications such as in direct anonymous attestation (DAA) [9] and anonymous electronic identity (eID) token [12, 21]; and implementations such as U-prove [21], Idemix [12] and in java cards [5]. *Revocation* is indispensable in credential systems in practice, as dispute, compromise, mistake, identity change, hacking and insecurity could make any credential become invalid before its expiration. The anonymity requirement makes revocation for anonymous credentials more challenging, as the user also needs to anonymously prove that her credentials are not revoked. The primary revocation method so far is to use accumulator [23, 2], as the cost for each proof of not being revoked is constant. An anonymous credential system is *delegatable* [3] if its credential could be delegated from one user to another user so that a user could anonymously prove a credential which is delegated some levels away from the original issuer. Delegation is important for efficiently managing any kind of organizations, as one person can not do everything and should delegate some authority to colleagues or subordinates to execute her tasks.

Our contribution.
We introduce and formally define a new notion of *homomorphic proof*, which means there is an operation which 'adds' some proofs, their statements and their witnesses to get a new valid proof of the 'sum' statement and the 'sum' witness. We present and prove a construction for homomorphic proofs from GS proofs [18]. GS proof's high level of generalization could partly explain its large number of applications, such as group signatures, ring signatures, mix-nets, anonymous credentials, oblivious transfer. Our homomorphic construction aims to use the most general form of GS proofs to maximize the range of possible applications.

As a signature or authentication could be viewed as a proof of authenticity on data, some more straight directions for applying homomorphic proofs could be homomorphic signatures [19] and homomorphic authentication [1], which have found applications in provable cloud storage [1], network coding [13, 24], digital photography [20] and undeniable signatures [22]. Homomorphic encryption and commitment schemes have been used in mix-nets , voting [15], anonymous credentials [3] and other multi party computation systems. Independently from us, homomorphic NIZK is proposed in [14] and used for homomorphic encryption. Gentry's recent results on fully homomorphic encryption [16] allow computing any generic functions of encrypted data without decryption and could be applied in scenarios such as cloud computing and searchable encryption. There is a chance that homomorphic proofs could also be useful in these contexts. In this paper, we will look at its application to *blacklisting delegatable anonymous credentials*.

Blacklisting anonymous credentials has been relying on accumulators. Identities of revoked credentials are accumulated in a blacklist and a user proves that her credential is not revoked by using the accumulator's NM proof, whose cost is constant, to prove that the credential's identity is not accumulated. For delegatable credentials, when a credential is revoked, a natural rule is to consider all delegated descendants of the credential to be revoked. Applying that rule in delegatable anonymous credentials, a user must be able to anonymously prove that all ancestor credentials of her credential are not revoked, even when the blacklist changes.

So the main difficult challenge is to create a new type of accumulators for blacklisting delegatable anonymous credentials that satisfies the following requirements. First, user A, without leaking private information, could delegate the ability to prove that her credential's identity is not accumulated in any blacklist to user B so that such proofs generated by A and B are indistinguishable and the blacklist could change anytime. Second, the delegation should be unlinkable, i.e. it should be hard to tell if two such delegations come from the same delegator A. Third, user B should be 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, when receiving some delegation information, one should be able to verify that it is correctly built. We call such a scheme as *accumulator with delegatable NM proofs (ADNMP)*.

With homomorphic proofs, we propose the *first* solution to the above challenge, i.e. constructing an ADNMP scheme, and then using it to revoke delegatable anonymous credentials. We define a new model for accumulators and extend it to define security requirements for delegatable NM proofs. We prove security of the accumulator scheme and the delegatable anonymous credentials with revocation system. Their constructions in the SXDH (Symmetric External Diffie Hellman) or SDLIN (Symmetric Decisional Linear) instantiations of GS proofs allow using the most efficient curves for pairings [17].

Homomorphic proofs bring delegatability of proofs to another level. A proof's *statement* often consists of some *commitments* of variables (witnesses) and some *conditions*. In [3], a proof could be randomizable or malleable, that means it is possible to generate a new proof and to randomize the statement's commitments without 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 proofs, which are homomorphic. By linearly combining these proofs and their statements, the delegatee could generate several new proofs for several other statements with different conditions. In delegatable NM proofs of accumulators, changing blacklist is an example of changing a statement's conditions. In short, the BCCKLS paper [3] deals with delegating proofs of the same statements' conditions. This paper deals with delegating proofs of changing statements' conditions.

Our final contribution is an implementation of RAC, a system for Revocation of Anonymous Credentials. Its core component is the ADNMP scheme. RAC could be used for several anonymous credential systems [21, 10, 4, 3]with or without delegatability, some of which have been implemented or commercialized. Previous discussions on accumulators always focus on the constant costs of their proofs. But there is a tradeoff: the cost of computing and updating a witness is linear to the number of accumulated elements. So in scenarios which require lots of changes in the accumulated set and only a few proofs, accumulators may be an inefficient choice or must be adjusted to optimize performance. This paper will show such an optimization for RAC's design.

The next section recalls some background knowledge and the following sections present our results in homomorphic proofs, ADNMP, blacklisting delegatable anonymous credentials, and RAC.

2. BACKGROUND

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. Denote $\leftarrow$ for random output. For a group $\mathbb{G}$ with identity $\mathcal{O}$, denote $\mathbb{G}^* := \mathbb{G}\backslash\{\mathcal{O}\}$. $Mat_{m\times n}(\mathcal{R})$ is the set of matrices with size $m \times n$ of elements in $\mathcal{R}$. For a matrix Γ, denote $\Gamma[i,j]$ the value at row i_{th} and column j_{th}. A vector $\vec{z}$ of l elements can be seen as a matrix of l rows and 1 column. For a vector or tuple z, denote $z[i]$ the i^{th} element. Notations of algorithms may omit inputs, such as public parameters *Para*, when appropriate.

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 [18]. 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 $\mathbb{G}_2$.

2.1 Non-Interactive 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. Completeness means that for every PPT adversary $\mathcal{A}$, $\Pr[Para \leftarrow \mathsf{Setup}(1^k); (Sta, Wit) \leftarrow \mathcal{A}(Para); Proof \leftarrow \mathsf{Prove}(Para, Sta, Wit) :$ $\mathsf{Verify}(Para, Sta, Proof) = 1 \text{ if } (Para, Sta, Wit) \in \mathbf{R}]$ is overwhelming. Soundness means that for every PPT adversary $\mathcal{A}$, $\Pr[Para \leftarrow \mathsf{Setup}(1^k); (Sta, Proof) \leftarrow \mathcal{A}(Para) :$ $\mathsf{Verify}(Para, Sta, Proof) = 0 \text{ if } (Para, Sta, Wit) \notin \mathbf{R}, \forall Wit]$ 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 [18] 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 [3] has another PPT algorithm RandProof, that takes as input (*Para*, *Sta*, *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* [3] 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* [3] 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* [4] for a bijection F if there is a PPT extractor (ExSet, ExWit) such that ExSet's output *Para* is distributed identically to Setup's output, and for every PPT adversary $\mathcal{A}$, $\Pr[(Para, td) \leftarrow \mathsf{ExSet}(1^k); (Sta, Proof) \leftarrow \mathcal{A}(Para); Ext \leftarrow \mathsf{ExWit}(td, Sta, Proof) : \mathsf{Verify}(Para, Sta, Proof) = 1 \wedge (Para, Sta, F^{-1}(Ext)) \notin \mathbf{R}]$ is negligible. As in [4], 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*: $Proof \leftarrow \mathsf{NIPK}[x_1 \text{ in } C_1, ..., x_k \text{ in } C_k]\{F(Para, Wit) : Condition(Para, Wit)\}$.

2.2 Groth-Sahai (GS) Proofs

This general description of GS proofs is based on the GS full version paper [18] which is updated and free from previous errors [17].

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 $a_1, \ldots, a_m \in A_1$, $b_1, \ldots, b_n \in A_2$ and $t \in A_T$. For any matrix $\Gamma \in Mat_{m \times n}(\mathcal{R})$, 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 $\vec{u_2} \in B_2^{\hat{n}}$, com-

mitments 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 could 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 $\mathbb{G}_T$ could 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 $\mathbb{G}_2$.
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 $\vec{a}\cdot\vec{y} + \vec{x}\cdot\vec{b} + \vec{x}\cdot\Gamma\vec{y} = t$. 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 [18], in the SXDH and Decisional Linear (DLIN) [17] instantiations, for statements consisting of only multi-scalar multiplication and quadratic equations, GS proofs are composable ZK.

3. HOMOMORPHIC PROOFS

3.1 Formalization

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

DEFINITION. *Let* (Setup, Prove, 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 said to be 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 said to be a set of* ***strongly homomorphic*** *proofs if* $(\Pi, +_\Pi, I_\Pi)$ *forms an abelian group where* I_Π *is the identity element.*

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 addition of n times of $(Sta, Wit, Proof)$. As such, we also use $\sum_i a_i(Sta_i, Wit_i, Proof_i)$ 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 (section 2.2). Each map $\iota_1 : A_1 \to B_1$ satisfies $\iota_1(x_1 + x_2) = \iota_1(x_1) + \iota_1(x_2)$, $\forall x_1, x_2 \in A_1$, and similarly for ι_2.

We first define the **identity** $I_{GS} = (Sta_0, Wit_0, Proof_0)$. Sta_0 consists of L GS equations $(\vec{a}_0, \vec{b}_0, \Gamma_0, t_0)$, 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:

$\diamond$ Let m be the dimension of $\vec{b}_0$, $\vec{x}_0$ and $\vec{c}_0$. There exists a set $M \subseteq \{1, ..., m\}$ 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 $\bar{M} := \{1, ..., m\}\backslash M$.

$\diamond$ Let n be the dimension of $\vec{a}_0$, $\vec{y}_0$ and $\vec{d}_0$. There exists a set $N \subseteq \{1, ..., n\}$ 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 $\bar{N} := \{1, ..., n\}\backslash N$.

$\diamond$ Both $(\forall i \in \bar{M}, \forall j \in \bar{N})$ and $(\forall i \in M, \forall j \in N)$: $\Gamma_0[i, j] = 0$.

$\diamond$ $t_0 = 0$, $\vec{\pi}_0 = 0$, 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:

$\diamond$ $\forall i \in M$, $x[i] = x_0[i]$ and $c[i] = c_0[i]$. $\forall j \in \bar{M}$, $b[j] = b_0[j]$.

$\diamond$ $\forall i \in N$, $y[i] = y_0[i]$ and $d[i] = d_0[i]$. $\forall j \in \bar{N}$, $a[j] = a_0[j]$.

$\diamond$ 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.

$\diamond\ \forall i \in M$: $x[i] := x_1[i]$; $c[i] := c_1[i]$; $b[i] := b_1[i] + b_2[i]$. $\forall j \in \bar{M}$: $b[j] := b_1[j]$; $x[j] := x_1[j] + x_2[j]$; $c[j] := c_1[j] + c_2[j]$.

$\diamond\ \forall i \in N$: $y[i] := y_1[i]$; $d[i] := d_1[i]$; $a[i] := a_1[i] + a_2[i]$. $\forall j \in \bar{N}$: $a[j] := a_1[j]$; $y[j] := y_1[j] + y_2[j]$; $d[j] := d_1[j] + d_2[j]$.

$\diamond$ If $(i \in \bar{M}) \vee (j \in \bar{N})$, then $\Gamma[i,j] := \Gamma_1[i,j]$. Otherwise, $\Gamma[i,j] := \Gamma_1[i,j] + \Gamma_2[i,j]$.

$\diamond\ t = t_1 + t_2$, $\vec{\pi} = \vec{\pi}_1 + \vec{\pi}_2$, and $\vec{\psi} = \vec{\psi}_1 + \vec{\psi}_2$.

THEOREM 3.1. *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 3.1 could be found in the Appendix.

4. ACCUMULATOR

4.1 Model

An *universal accumulator* consists of the following PPT algorithms.

$\diamond$ 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.*

$\diamond$ 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 [2].

$\diamond$ A 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 $AcVal$.

$\diamond$ An **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. As this paper does not deal with membership proofs, we refer to an universal accumulator as (Setup, ProveNM, VerifyNM, CompNMWit, Accu).

4.2 Delegatable NM Proofs for Accumulators

Delegating ability to prove statements is to allow someone else to prove the statements on one's behalf without revealing the witness, even if the statements' conditions are changing over time. For privacy reasons, adversaries could not distinguish different delegations coming from different users. Moreover, the delegatee could verify a delegation and unlinkably redelegate the proving ability further to other users.

Therefore, delegating an accumulator's NM proofs should meet 4 conditions, as formalized in Definition 4.1. *Delegatability* means that an element Ele's owner can delegate her ability to prove that Ele is not accumulated without simply revealing Ele. Even if the set of accumulated elements is changing overtime, the delegatee does not need to contact the delegator again to generate the proof. The owner does it by giving the delegatee a key De generated from Ele. The proof generated from De by CompProof is indistinguishable from a proof generated by ProveNM. *Unlinkability* means that a delegatee should not be able to distinguish whether or not 2 delegating keys originating from the same element. It implies that it is computationally hard to compute an element from its delegating keys. *Redelegatability* means that the delegatee could redelegate De as De' to other users, so that the distributions of De and De' are indistinguishable. *Verifiability* means that one should be able to validate that a delegating key De is correctly built.

DEFINITION 4.1. *An universal accumulator (*Setup, ProveNM, VerifyNM, CompNMWit, Accu*) provides delegatable NM proofs if there exist PPT algorithms: delegating* Dele, *redelegating* Rede, *validating* Vali *and computing proof* CompProof *satisfying.*

$\diamond$ *Delegatability: For every PPT algorithm $(\mathcal{A}_1, \mathcal{A}_2)$, $|Pr[$ $(Para,\ Aux) \leftarrow$ Setup(1^k); $(Ele, AcSet, state) \leftarrow \mathcal{A}_1(Para)$; $AcVal \leftarrow$ Accu$(Para,\ AcSet)$; $Wit \leftarrow$ CompNMWit$(Para,\ Ele,\ AcSet,\ AcVal)$; $Proof_0 \leftarrow$ ProveNM$(Para,\ AcVal,\ Wit)$; $De \leftarrow$ Dele$(Para,\ Ele)$; $Proof_1 \leftarrow$ CompProof$(Para,\ De,\ AcSet,\ AcVal)$; $b \leftarrow \{0,1\}$; $b' \leftarrow \mathcal{A}_2(state,\ AcVal,\ Wit,\ De,\ Proof_b)$: $(Ele \notin AcSet) \wedge b = b'] - 1/2|$ is negligible.*

$\diamond$ *Unlinkability: For every PPT algorithm $\mathcal{A}$, $|Pr[$ $(Para,\ Aux) \leftarrow$ Setup(1^k); $(Ele_0, Ele_1) \leftarrow Dom_{Para}$; $De \leftarrow$ Dele $(Para,\ Ele_0)$; $b \leftarrow \{0,1\}$; $De_b \leftarrow$ Dele$(Para, Ele_b)$; $b' \leftarrow \mathcal{A}(Para,\ De,\ De_b)$: $b = b'] - 1/2|$ is negligible.*

$\diamond$ *Redelegatability: For every PPT algorithms $(\mathcal{A}_1, \mathcal{A}_2)$, $|Pr[$ $(Para,\ Aux) \leftarrow$ Setup(1^k); $(Ele, state) \leftarrow \mathcal{A}_1(Para)$; $De \leftarrow$ Dele$(Para, Ele)$; $De_0 \leftarrow$ Dele$(Para, Ele)$; $De_1 \leftarrow$ Rede $(Para,\ De)$; $b \leftarrow \{0,1\}$; $b' \leftarrow \mathcal{A}_2(state, De, De_b) : b = b'] - 1/2|$ is negligible.*

$\diamond$ *Verifiability: For every PPT algorithm $\mathcal{A}$, $|Pr[$ $(Para,\ Aux) \leftarrow$ Setup(1^k); $Ele \leftarrow \mathcal{A}(Para)$; $De \leftarrow$ Dele$(Para,\ Ele)$: Vali$(Para, De) = 1$ if $Ele \in Dom_{Para}] - 1|$ and $|Pr[(Para, Aux) \leftarrow$ Setup(1^k); $De' \leftarrow \mathcal{A}(Para) :$ Vali$(Para,\ De') = 0$ if $De' \notin \{De | De \leftarrow$ Dele$(Para, Ele')$; $Ele' \in Dom_{Para}\}] - 1|$ are negligible.*

On the other hand, given an element Ele', the delegatee can accumulate Ele' and try to prove that Ele is not accumulated using De. If she can't prove that anymore, she can conclude that $Ele \equiv Ele'$. So for any ADNMP, given an element Ele and a delegating key De, one can tell if De is generated by Ele. Due to this restriction, in the accumulator's applications, Ele should be a secret that only its owner or a trusted authority knows.

5. AN ADNMP SCHEME

We propose a dynamic universal ADNMP. Its Setup, Accu and UpdateVal are generalized from those in [2].

◇ Setup: We need GS instantiations where GS proofs for this accumulator are composable ZK. As its GS proofs are only for multi-scalar or quadratic equations, we could use either the SXDH or SDLIN [17] instantiations, as explained in section 2. This paper will 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 as in section 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 which hasn't accumulated q elements and update $V_j' = (\delta + a')V_j$; if such V_j couldn't be 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 $V_j' = 1/(\delta + a')V_j$.

REMARKS. Previous accumulators [2] require that q of ς is the upper bound on the number of elements to be accumulated, i.e. $m = 1$. We could relax this requirement by the above generalization which allows this ADNMP to work no matter whether or not q is less than the number of accumulated elements. It also allows q to be set up smaller.

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 $Poly(\delta)$ mod $(\delta + y_2)$ in Z_p, and X_{j1} be scalar product of the quotient and P_1. Similar to [2], 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 [8], though it is not clear which assumption is stronger. It is in the extended uber-assumption family [7] 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 $(\frac{y_3}{\delta + y_2} P_1, y_2, y_3 P_2, y_3 A)$ where $y_3 \neq 0$.*

We will show later that if one could prove the knowledge of $(y_{j3}P_2, y_{j3}A)$ satisfying $(\delta + y_2)X_{j1} + y_{j3}P_1 = V$ and y_2 is accumulated in V but $y_{j3} \neq 0$, then she could 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.

◇ CompNMWit takes in 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 [2] 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 $T_j \neq \mathcal{O}$, $\forall j$. It accepts if both of them pass or rejects otherwise.

THEOREM 5.1. *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).*

Proof sketch of theorem 5.1 could be found in the Appendix.

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
$\bigwedge_{j=1}^m ((\delta + y_2)X_{j1}^{(2)} + y_{j3}^{(2)}P_1 = V_j^{(2)} \wedge X_{j3}^{(2)} - y_{j3}^{(2)}A = 0 \wedge y_{j3}^{(2)}X_{j2} = T_j^{(2)})$ will 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 ideas of constructing the accumulator's delegatable NM proof. We see that a component accumulator value $V = \prod_{i=1}^{k}(\delta + a_i)\delta P_1$ of $\{a_1, ..., a_k\}$ can be written as $V = \sum_{i=0}^{k} b_i \delta^{k+1-i} P_1$ where $b_0 = 1$ and $b_i = \sum_{1 \le j_1 < j_2 < ... < j_i \le k} \prod_{l=1}^{i} a_{j_l}$, that means V can be written as a linear combination of $\delta P_1, \ldots, \delta^{k+1} P_1$ in ς.
We can construct proofs, which are homomorphic, for each $(\delta+y_2)X_1^{(i)}+y_3^{(i)}P_1 = \delta^i P_1 \wedge X_3^{(i)} - y_3^{(i)} A = 0 \wedge y_3^{(i)} X_2 = T^{(i)}$ where $i \in \{1, ..., k+1\}$. Using the same linear combination of $\delta P_1, \ldots, \delta^{k+1} P_1$ for V, we can 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 $T = \sum_{i=0}^{k} b_i T^{(k+1-i)}$. We now provide the algorithms for delegating NM proofs and its security theorem. We also add UpdateProof to be used in place of CompProof when possible for efficiency.

$\diamond$ 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 $\bigwedge_{i=1}^{q+1}((\delta + y_2)X_1^{(i)} + y_3^{(i)} P_1 = \delta^i P_1 \wedge X_3^{(i)} - y_3^{(i)} A = 0 \wedge y_3^{(i)} X_2 = T^{(i)})$.

$\diamond$ 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'.

$\diamond$ 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.

$\diamond$ CompProof$(Para, De, AcSet, AcVal)$. 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.

$\diamond$ 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 CompProof. 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 [7], but can be proved in generic groups similarly to the PowerDDH assumption [11]. Proof sketch of theorem 5.2 could be found in the Appendix.

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 $B_0, x_0 B_0, \ldots, x_0^q B_0, B_1, x_b B_1, \ldots, x_b^q B_1$, no PPT algorithm could output $b' = b$ with a probability non-negligibly better than a random guess.*

THEOREM 5.2. *The accumulator provides delegatable NM proofs, based on ESDH, DSDH and the assumption underlying the GS instantiation (SXDH or SDLIN).*

6. REVOKING DELEGATABLE ANONYMOUS CREDENTIALS

6.1 Model

This is a model of delegatable anonymous credential with revocation systems. For each credential proof, a user uses a new nym which is indistinguishable from her other nyms. We need another type of nyms for revocation called *r-nym* to distinguish between these 2 types of nyms. When an r-nym is revoked, its owner cannot prove credentials anymore. Participants include users and a Blacklist Authority (BA) owning a blacklist BL which is empty initially. The PPT algorithms are:

$\diamond$ Setup(1^k) outputs trusted public parameters $Para_{DC}$, BA's secret key Sk_{BA}, and an initially empty blacklist BL.

$\diamond$ KeyGen$(Para_{DC})$ outputs a secret key Sk and a secret r-nym Rn for a user.

$\diamond$ NymGen$(Para_{DC}, Sk, Rn)$ outputs a new nym Nym with an auxiliary key $Aux(Nym)$.

$\diamond$ An user O becomes 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.

$\diamond$ Issue$(Para_{DC}, Nym_O, Sk_I, Rn_I, Nym_I, Aux(Nym_I), Cred, DeInf, Nym_U, L) \leftrightarrow$ Obtain$(Para_{DC}, Nym_O, Sk_U, Rn_U, Nym_U, Aux(Nym_U), Nym_I, L)$ 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 $Cred_U$. $DeInf$ 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 $DeInf = 1$ (optional).

$\diamond$ Revoke$(Para_{DC}, Sk_{BA}, Rn, BL)$ updates BL so that a revoked user Rn cannot prove credentials or delegate.

$\diamond$ 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.

$\diamond$ 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 [3] are the introductions of BA with Sk_{BA} and BL; r-nyms; delegation information $DeInf$; Revoke; and 2 $CredProof$'s conditions (ii) and (iii).

Delegability and Anonymity. They 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 (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*. Otherwise, she should know that such delegation could compromise her anonymity when issuing. 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 Issue $\leftrightarrow$ Obtain. Other privacy properties, such as anonymity of $CredProof$, Nym and the delegatee, are still maintained.
This limitation will be reflected in the Anonymity definition and is related to the restriction on ADNMP mentioned in section 4.2. 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'.

Exposing R-nyms. We can use one of previous methods for BA to obtain r-nyms to revoke. There could be an authority who could force any user to reveal his r-nym to BA and prove his ownership 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. If such an enforcement is difficult, another method adopted from group signatures [6] is an Opening Authority who can open any disputed $CredProof$ to find its generator's r-nym. The third option is a Nym Authority who controls users' r-nyms and makes requests to BA to revoke r-nyms.

6.2 Security

The full definition could be found in this paper's full version. There are 3 requirements which are extended from the security definition of delegatable anonymous credentials [3].

Correctness: 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.

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 simulators, including SimSetup, SimProve, SimObtain and SimIssue. The additions to the definition in [3] 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 simulators. 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.

Unforgeability: 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 [3] include the followings. Oracle maintains a list of honest parties, which may or may not include BA. 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.

6.3 A scheme

Overview. We first describe intuitions of the BCCKLS delegatable anonymous credential scheme [3] and 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 $\mathsf{VerifyAuth}(Para_{At}, Sk, \vec{m}, Auth)$ accepts if and only if $Auth$ validly authenticates $\vec{m}$ under Sk. The scheme could be F-*Unforgeable* [4] for a function F, which means $(F(\vec{m}), Auth)$ is unforgeable without obtaining an authenticator on $\vec{m}$; or *certification secure*, which means no PPT adversary, even after obtaining an authenticator by the challenge secret key, can forge another authenticator. Besides, BCCKLS uses a protocol (AuthPro) for an user to obtain from an issuer an NIZKPK of an authenticator on $\vec{m}$ without revealing anything about $\vec{m}$.
An user U could 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, ... , 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 could extend 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
$CredProof \leftarrow \mathsf{NIZKPK}[Sk_O \text{ in } Nym_O[1], Sk \text{ in } Nym[1],$
$Rn \text{ in } Nym[2]]\{(F(Sk_O), F(Sk_1), F(Rn_1), ..., F(Sk_L),$
$F(Rn_L), F(Sk), F(Rn), auth_1, ..., auth_L, auth_{L+1}) :$
$\mathsf{VerifyAuth}(Sk_O, (Sk_1, Rn_1, r_1), auth_1) \wedge (Rn_1 \notin BL) \wedge$
$\mathsf{VerifyAuth}(Sk_1, (Sk_2, Rn_2, r_2), auth_2) \wedge (Rn_2 \notin BL) \wedge ... \wedge$
$\mathsf{VerifyAuth}(Sk_{L-1}, (Sk_L, Rn_L, r_L), auth_L) \wedge (Rn_L \notin BL) \wedge$
$\mathsf{VerifyAuth}(Sk_L, (Sk, Rn, r_{L+1}), auth_{L+1}) \wedge (Rn \notin BL)\}$ (1).

Description. The building blocks consist of: (i) Those from BCCKLS, including $\mathcal{AU}$; AuthPro; H; and a malleable NIPK credential proof system (CredPS) of PKSetup, PKProve, PKVerify, RandProof, with commitment Com. (ii) An accumulator with a randomizable delegatable NM proof sytem (NMPS) of AcSetup, ProveNM, VerifyNM, CompNMWit, Accu, Dele, Rede, Vali, CompProof, with commitment ComNM. (iii) A randomizable proof system (EQPS), whose setup consists of PKSetup and AcSetup, to prove that 2 given commitments by Com and ComNM commit to the same value.
Assume a delegating key De contains a commitment of its element Ele. CompProof and Rede generate Ele's commitment in their outputs by randomizing the commitment in De. Elements of the accumulator domain and the authenticator's keyspace can be committed by Com.

The BCCKLS building blocks could be instantiated as in [3]; an ADNMP instantiation is presented in section 4; and an EQPS instantiation with composable ZK can be constructed from [4]. They all share the same bilinear pairing parameters, so elements of the accumulator domain and the authenticator's keyspace are in $\mathbb{Z}_p$ and committable by Com. The concatenation of instantiated CredPS, NMPS and EQPS forms a GS proof system and thereby is randomizable, partially extractable, and composable ZK. The following algorithm inputs are the same as in the model and omitted.

$\diamond$ Setup: Use $\mathsf{PKSetup}(1^k)$, $\mathsf{AtSetup}(1^k)$ and $\mathsf{AcSetup}(1^k)$ 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 set and its initial accumulator value, $Para_{DC} = (Para_{PK}, Para_{At}, Para_{Ac}, H)$, and $Sk_{BA} = Aux_{Ac}$.

$\diamond$ KeyGen: Run $\mathsf{AuthKg}(Para_{At})$ to output a secret key Sk. Output a random r-nym Rn from the accumulator's domain.

$\diamond$ NymGen: Generate random $Open_{Sk}$ and $Open_{Rn}$, and output nym $Nym = (\mathsf{Com}(Sk, Open_{Sk}), \mathsf{Com}(Rn, Open_{Rn}))$ and $Aux(Nym) = (Open_{Sk}, Open_{Rn})$.

$\diamond$ The credential originator O publishes a Nym_0 and a proof $NMProof_0$ that Rn_O is not revoked that O has to update when BL changes.

$\diamond$ Issue $\leftrightarrow$ Obtain: If $L = 0$ and $Nym_O \neq Nym_I$, aborts. Issuer I verifies that Nym_I and $Cred$ are valid with Sk_I, Rn_I and $Aux(Nym_I)$, and user U verifies that Nym_U is valid with Sk_U, Rn_U and $Aux(Nym_U)$. After that, they both compute $r_{L+1} = H(Nym_O, atributes, L+1)$ for a set of attributes for that level's credential, as in [3]. They then run AuthPro for U to receive: $Proof_U \leftarrow \mathsf{NIZKPK}[Sk_I \text{ in } Nym_I[1], Sk_U \text{ in } \mathsf{Com}(Sk_U, 0), Rn_U \text{ in } \mathsf{Com}(Rn_U, 0)]$
$\{(F(Sk_I), F(Sk_U), F(Rn_U), auth) : \mathsf{VerifyAuth}(Sk_I, (Sk_U, Rn_U, r_{L+1}), auth)\}$.
U's output is $Cred_U = Proof_U$ when L = 0. Otherwise, suppose the users on 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 for $j \in \{1, ..., L\}$, Sk_j and Rn_j are authenticated by Sk_{j-1} (with r_j). Concatenate $Proof_U$ and $CredProof_I$ and project 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 $j \in \{1, ..., L-1\}$. When $DeInf$ is in the input, I Redes these delegating keys, updates and randomizes $EQProof_j$ to match the new 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 U.

$\diamond$ Revoke: Add Rn to the accumulated set and update the accumulator value.

$\diamond$ CredProve: Abort if $Nym \neq (\mathsf{Com}(Sk, Open_{Sk}), \mathsf{Com}(Rn, Open_{Rn}))$. Use ProveNM to generate a proof $NMProof$ that Rn is not blacklisted. Generate $EQProof$ 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 $EQProof$ to get $CredProof'$. If the optional $DeInf$ is omitted, just output $CredProof'$. Otherwise, use CompProof to generate a proof $NMChainProof$ that each Rn_j's on the user's chain of delegators is not accumulated in the blacklist. Update and randomize $EQProof_j$ for $j \in \{1, ..., L\}$ to match with $NMChainProof$ and $CredProof'$. Concatenate $NMChainProof$ and $CredProof'$ to output $CredProof$ as described in (1).

$\diamond$ CredVerify runs PKVerify, VerifyNM and verifies $EQProof$s to output accept or reject.

THEOREM 6.1. *If the authentication scheme is F-unforgeable and certification-secure; a concatenation of CredPS, NMPS and EQPS is randomizable, partially extractable, and composable ZK; and H is collision resistant, then this construction is a secure revocable delegatable anonymous credential system.*

Proof sketch of theorem 6.1 could be found in the Appendix.

7. RAC

RAC (Revocation of Anonymous Credentials) is developed in C++ and includes 2 libraries. The first implements the ADNMP scheme in the SXDH instantiation proposed in section 4, and could be used to develop accumulator's applications. It provides API for ADNMP's algorithms AcSetup, ProveNM, VerifyNM, CompNMWit, Accu, Dele, Rede, Vali and CompProof. The second library depends on the ADNMP library to perform revoking anonymous credentials as follows. A blacklist authority could use it to create a blacklist and accumulate revoked identities of anonymous credentials. A user could prove that an identity is not accumulated in a blacklist, and to delegate that proof. Another user could redelegate or compute proofs from the delegation. Several anonymous credential systems based on prime order with or without delegatability [21, 10, 4, 3] could be integrated with RAC for revocation. Besides the BCCKLS system [3], a non-delegatable example is U-Prove [21], a commercialized anonymous credential system developed by Microsoft, that has been released for community technology preview.

RAC is the first solution for revoking **delegatable** anonymous credentials, and the first implementation of an universal accumulator. The only previous prime-order universal accumulator [2] requires Random Oracle (RO) for non-interactive proofs and its prover needs 3 pairings, whereas RAC requires no RO and its prover does no pairing. For RAC's accumulator with $q = 500$, using 256-bit BN pairing curves, on a regular 2.4 GHz Intel 2 Core with 4 GB RAM, ProveNM takes 0.14 s and Dele takes 69.38 s.

PERFORMANCE OPTIMIZATION. In a system using this accumulator, depending on the accumulator's workloads, a central entity (who knows Aux) can always adjust the value of q at any time to optimize efficiency. We utilize this advantage in the following exemplified scenario. Based on the work loads on the accumulator's operations and the number of accumulated elements, we look for the optimal value of q. Assume in an application, the number of accumulated elements is around a constant Q overtime and elements could be added or removed. Let $m = \lceil Q/q \rceil$, let the computation unit be a scalar product, and let the approximate costs of the accumulator's operations be as follows (generalized for both SXDH and SDLIN instantiations): Accu - m; UpdateVal - 1; CompNMWit - $(mq \approx Q)$; $UpdateWitness$ - 2; ProveNM - $\alpha_1 m$; VerifyNM - $\alpha_2 m$; Dele - $\beta_1 q$; Rede - $\beta_2 q$; Vali - $\beta_3 q$; CompProof - $(\beta_4 mq + \alpha_3 m)$; UpdateProof - $(\beta_4 q + \alpha_3 m)$; where α_i and β_i are constants. For simplicity and analysis of a common user, we will ignore operations which are rare or mostly performed by a central entity, or whose cost does not change when q changes.

Suppose over a period of time, say 1 year, the average numbers of runs of operations per user are as follows: ProveNM - a_1; VerifyNM - a_2; Dele - b_1; Rede - b_2; Vali - b_3; CompProof - a_3; UpdateProof - c; where a_i, b_i and c are constants. The total cost per user per period is $S(q) \approx a_1\alpha_1 m + a_2\alpha_2 m + b_1\beta_1 q + b_2\beta_2 q + b_3\beta_3 q + a_3\alpha_3 m + a_3\beta_4 mq + c\beta_4 q + c\alpha_3 m \approx (a_1\alpha_1 + a_2\alpha_2 + a_3\alpha_3 + c\alpha_3)Q/q + (b_1\beta_1 + b_2\beta_2 + b_3\beta_3 + c\beta_4)q + a_3\beta_4 Q$.

As $(\sqrt{(a_1\alpha_1 + a_2\alpha_2 + a_3\alpha_3 + c\alpha_3)Q/q} - \sqrt{(b_1\beta_1 + b_2\beta_2 + b_3\beta_3 + c\beta_4)q})^2 \geq 0$, we have $(a_1\alpha_1 + a_2\alpha_2 + a_3\alpha_3 + c\alpha_3)Q/q + (b_1\beta_1 + b_2\beta_2 + b_3\beta_3 + c\beta_4)q \geq 2\sqrt{(a_1\alpha_1 + a_2\alpha_2 + a_3\alpha_3 + c\alpha_3)(b_1\beta_1 + b_2\beta_2 + b_3\beta_3 + c\beta_4)Q}$.

So minimum of $S(q)$ happens when $q \approx min(Q, \sqrt{\frac{a_1\alpha_1 + a_2\alpha_2 + a_3\alpha_3 + c\alpha_3}{b_1\beta_1 + b_2\beta_2 + b_3\beta_3 + c\beta_4}}Q)$.

So intuitively, if users have to generate proofs lots more than delegations, then $q = Q$. If their cost amounts are about the same, then $q \approx \sqrt{Q}$.

8. REFERENCES

[1] G. Ateniese, S. Kamara, and J. Katz. Proofs of storage from homomorphic identification protocols. In *ASIACRYPT 2009*, 2009.

[2] M. H. Au, P. P. Tsang, W. Susilo, and Y. Mu. Dynamic universal accumulators for DDH groups and their application to attribute-based anonymous credential systems. In *CT-RSA 2009*, 2009.

[3] M. Belenkiy, J. Camenisch, M. Chase, M. Kohlweiss, A. Lysyanskaya, and H. Shacham. Randomizable proofs and delegatable anonymous credentials. In *CRYPTO 2009*, 2009.

[4] M. Belenkiy, M. Chase, M. Kohlweiss, and A. Lysyanskaya. P-signatures and noninteractive anonymous credentials. In *TCC 2008*, 2008.

[5] P. Bichsel, J. Camenisch, T. Groß, and V. Shoup. Anonymous credentials on a standard java card. In *ACM CCS 09*, 2009.

[6] D. Boneh, X. Boyen, and H. Shacham. Short group signatures. In *CRYPTO 2004*, 2004.

[7] X. Boyen. The uber-assumption family (invited talk). In *PAIRING 2008*, 2008.

[8] X. Boyen and B. Waters. Full-domain subgroup hiding and constant-size group signatures. In *PKC 2007*, 2007.

[9] E. F. Brickell, J. Camenisch, and L. Chen. Direct anonymous attestation. In *ACM CCS 04*, 2004.

[10] J. Camenisch and A. Lysyanskaya. Signature schemes and anonymous credentials from bilinear maps. In *CRYPTO 2004*, 2004.

[11] J. Camenisch, G. Neven, and A. Shelat. Simulatable adaptive oblivious transfer. In *EUROCRYPT 2007*,

2007.
[12] J. Camenisch and E. Van Herreweghen. Design and implementation of the idemix anonymous credential system. In *ACM CCS 02*, 2002.
[13] D. Charles, K. Jain, and K. Lauter. Signatures for network coding. In *International Journal on Information and Coding Theory*, 2006.
[14] Y. Dodis, K. Haralambiev, A. Lopez-Alt, and D. Wichs. Cryptography against continuous memory attacks, 2010.
[15] P.-A. Fouque, G. Poupard, and J. Stern. Sharing decryption in the context of voting or lotteries. In *FC 2000*, 2000.
[16] C. Gentry. Fully homomorphic encryption using ideal lattices. In *41st ACM STOC*, 2009.
[17] E. Ghadafi, N. Smart, and B. Warinschi. Groth sahai proofs revisited. In *PKC*, 2010.
[18] J. Groth and A. Sahai. Efficient non-interactive proof systems for bilinear groups. In C. ePrint 2007/155, editor, *EUROCRYPT 2008*, 2008.
[19] R. Johnson, D. Molnar, D. X. Song, and D. Wagner. Homomorphic signature schemes. In *CT-RSA 2002*, 2002.
[20] R. Johnson, L. Walsh, and M. Lamb. Homomorphic signatures for digital photographs. In *Suny Stony Brook*, 2008.
[21] Microsoft. U-prove community technology preview. In *https://connect.microsoft.com/*, 2010.
[22] J. Monnerat and S. Vaudenay. Generic homomorphic undeniable signatures. In *ASIACRYPT 2004*, 2004.
[23] P. P. Tsang, M. H. Au, A. Kapadia, and S. W. Smith. PEREA: towards practical TTP-free revocation in anonymous authentication. In *ACM CCS 08*, 2008.
[24] A. Yun, J. Cheon, and Y. Kim. On homomorphic signatures for network coding. In *Transactions on Computer*, 2009.

9. APPENDIX: PROOFS

Proof of theorem 3.1. 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 $i \in \{1,2\}$, $\vec{c_i} := \iota_1(\vec{x_i}) + R_i\vec{u_1}$, $\vec{d_i} := \iota_2(\vec{y_i}) + S_i\vec{u_2}$.

$$\vec{\pi_i} := R_i^\top \iota_2(\vec{b_i}) + R_i^\top \Gamma_i \iota_2(\vec{y_i}) + R_i^\top \Gamma_i S_i \vec{u_2} - T_i^\top \vec{u_2} + \sum_{j=1}^{\eta} r_j^{(i)} H_j \vec{u_2} \quad (1)$$

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

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

$$\vec{x} = \begin{pmatrix} \hat{X} \\ \tilde{X}_1 + \tilde{X}_2 \end{pmatrix}, \vec{b} = \begin{pmatrix} \hat{B}_1 + \hat{B}_2 \\ \tilde{B} \end{pmatrix}, R = \begin{pmatrix} \hat{R} \\ \tilde{R}_1 + \tilde{R}_2 \end{pmatrix}$$

$$\vec{c} = \begin{pmatrix} \hat{C} \\ \tilde{C}_1 + \tilde{C}_2 \end{pmatrix} = \begin{pmatrix} \iota_1(\hat{X}) + \hat{R}\vec{u_1} \\ \iota_1(\tilde{X}_1) + \tilde{R}_1\vec{u_1} + \iota_1(\tilde{X}_2) + \tilde{R}_2\vec{u_1} \end{pmatrix}$$
$$= \begin{pmatrix} \iota_1(\hat{X}) + \hat{R}\vec{u_1} \\ \iota_1(\tilde{X}_1 + \tilde{X}_2) + (\tilde{R}_1 + \tilde{R}_2)\vec{u_1} \end{pmatrix} = \iota_1(\vec{x}) + R\vec{u_1} \quad (3)$$

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

$$\vec{y} = \begin{pmatrix} \hat{Y} \\ \tilde{Y}_1 + \tilde{Y}_2 \end{pmatrix}, \vec{a} = \begin{pmatrix} \hat{A}_1 + \hat{A}_2 \\ \tilde{A} \end{pmatrix}, S = \begin{pmatrix} \hat{S} \\ \tilde{S}_1 + \tilde{S}_2 \end{pmatrix}$$

$$\vec{d} = \begin{pmatrix} \hat{D} \\ \tilde{D}_1 + \tilde{D}_2 \end{pmatrix} = \iota_2(\vec{y}) + S\vec{u_2} \quad (4)$$

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

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

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 $\pi = \pi_1 + \pi_2$

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

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

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

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 could 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*.

The other 4 conditions **Associativity**, **Commutativity**, **Identity element** and **Inverse element** of abelian groups can be easily validated. So the theorem holds.

Proof sketch of theorem 5.1. Its correctness and composable ZK comes from the GS proof and its instantiations and the fact that $y_2 \notin AcSet$ and $X_{j2} \neq 0$ means $T_j \neq \mathcal{O}$. And as described in [18], 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. Suppose an adversary could forge a 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 $(p,\mathbb{G}_1,\mathbb{G}_2,\mathbb{G}_T,e,P_1,\delta P_1,\ldots,\delta^{q+1}P_1,A,P_2,\delta P_2)$. Simulate random CRS σ with extracting trapdoor for GS proofs in either the SXDH or SDLIN instantiations so that from a commitment in $\mathbb{G}_2$ of $y \in Z_p$ and a commitment of $X \in \mathbb{G}_1$, we could respectively extract yP_2 and X. With the trapdoor, we could compute $\tau := \iota_2'(\delta)$ and we have all parameters for a simulated accumulator.

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 could 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 could find y_2. Suppose y_2 is accumulated in V_l which accumulates $\{a_1,...,a_k\}$. As $X_{l3} = y_{l3}A$, we could 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 could compute $\frac{y_{l3}}{y_1+y_2}P_1$ from X_{l1}, $\{a_1,...,a_k\}$ and ς. So now, we could find $(\frac{y_{l3}}{\delta+y_2}P_1, y_2, y_{l3}P_2, y_{l3}A)$ and break the assumption.

Proof sketch of theorem 5.2. To prove Delegatability, we see that CompProof's output is 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 ProveNM. Due to GS proofs' randomizability, the outputs has the same distribution which means Delegatability. For proving Redelegatability, we see that for the same y_2, the output $T'^{(i)}$, $i \in \{1,...,k+1\}$ of Rede has the same distribution as the output $T^{(i)}$, $i \in \{1,...,k+1\}$ of Dele. And for the same $T^{(i)}$, $i \in \{1,...,k+1\}$, Rede's output is a randomization of a proof that Dele could produce, so their outputs have the same distribution. Therefore, Dele and Rede output the same distribution that leads to Redelegatability. Verifiability comes from ESDH and the completeness and soundness of GS proofs, as De is a GS proof.

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). Consider 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 we can break the underlying assumption. If not, then we can break q-DSDH as follows. Suppose we are given a q-DSDH challenge $p,\mathbb{G}_1,\mathbb{G}_2,\mathbb{G}_T,e$, $P_1,P_2,B_0,x_0B_0,\ldots,x_0^qB_0,B_1,x_bB_1,\ldots,x_b^qB_1$. Simulate a CRS for GS proofs from the setup. Use the same simulation for GS proofs [18], simulate a proof De for $\bigwedge_{i=1}^{q+1}((\delta+y_2)X_1^{(i)} + y_3^{(i)}P_1 = \delta^iP_1 \wedge X_3^{(i)} - y_3^{(i)}A = 0 \wedge y_3^{(i)}X_2 = (-1)^ix_0^{i-1}B_0)$, and a proof De_b for $\bigwedge_{i=1}^{q+1}((\delta+y_2)X_1^{(i)} + y_3^{(i)}P_1 = \delta^iP_1 \wedge X_3^{(i)} - y_3^{(i)}A = 0 \wedge y_3^{(i)}X_2 = (-1)^ix_b^{i-1}B_1)$. 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 could tell with non-negligible advantage over a random guess if b is 0 or 1. That breaks q-DSDH.

Proof sketch of theorem 6.1. The scheme's correctness comes from correctness of its component authentication scheme and the concatenation of CredPS, NMPS and EQPS, and Delegatability and Redelegatability of the accumulator. The unforgeability proof is similar to the one in [3], based on F-unforgeability and certification-security of the authentication scheme, and partial extractability and soundness of the concatenation.

The anonymity proof is also similar to the one in [3], with the main difference is to create SimIssue indistinguishable from Issue with input $DeInf$. SimSetup includes AtSetup and the simulation setup SimConSetup for the concatenation. We see that the accumulator's 4 delegation properties still hold under parameters generated by SimConSetup, otherwise an adversary breaking one of the properties could distinguish SimConSetup and the concatenation setup ConSetup. We also see that a concatenation of just CredPS and EQPS is also composable ZK using simulation SimConSetup. SimIssue first generates a list of delegating keys for L random r-nyms. Based on the accumulator's Unlinkability and Redelegatability, the adversary can not distinguish this list from the list in $DeInf_U$ generated by Issue, as r-nyms of input $DeInf$ to Issue are also randomly generated and not revealed to the adversary. SimIssue then simulates the concatenation of CredPS and EQPS with r-nyms commitments in the delegating keys and merge it with the delegating keys to output. This output is indistinguishable from the output $(Cred_U, DeInf_U)$ generated by Issue.