Authenticated Multi-Proxy Accumulation Schemes for Delegated Membership Proofs

Hannes Salin[1], Dennis Fokin[2], and Alexander Johansson[3]

[1] Department of Information and Communication Technology, Swedish Transport Administration, Borlänge, Sweden hannes.salin@trafikverket.se
[2] fokin.dennis@gmail
[3] Saab AB, Stockholm, Sweden
alexander.johansson2@saab.com

Abstract. Proving ownership (or possibly non-ownership) of an attribute associated with an individual or device can be used in many different use cases. For a user to prove age, credibility, medical records or other type of attributes, cryptographic accumulators can be used. Also, in a federated authentication architecture, a user may prove such ownership via one or many proxies, e.g. a trusted party such as a bank or government institution. We propose an authenticated multi-proxy accumulation (AMPA) scheme for solving these types of scenarios without the need for encryption and still preserve privacy and remove data set leakage during set membership proving. We illustrate how an AMPA scheme easily can be constructed and present initial results from a proof of concept implementation.

Keywords: Proxy Accumulator · Proxy Signatures · Cryptography.

1 Introduction

Cloud computing and distributed data sharing is growing more than ever; outsourcing data storage to popular cloud solutions such as Amazon Web Services (AWS), Microsoft Azure and Google Cloud, which are the current top service providers [1], is constantly growing. However, government agencies and parts of the public sector are still reluctant to adopt cloud based solutions to third party service providers due to legal reasons (CLOUD Act [2]), but also due to security related issues [3]. At the same time, many types of information sharing use cases for individuals and government institutions remain, e.g. medical journal sharing between patient and hospital, and sharing and verification of criminal records between police authority, potential employers and other institutions. In all cases, the information could be considered attributes associated with an individual, i.e. the data owner. It is therefore of mutual interest that both the data owner and the data storage provider, e.g. the hospital or police authority, are able to verify the authenticity and integrity of the same data. Moreover, in some scenarios a (trusted) third party is the requester of verifying attribute data of a person, e.g. an employer needs to

verify an applicant's (non-existent) criminal records thus requesting proof from the police authorities. Ideally, these type of scenarios would benefit from using a cloud infrastructure shared between institutions and agencies, streamlined for fast data sharing and efficient real-time validity proof checking. However, due to GDPR it is problematic to share personal attributes between institutions, thus a mechanism for secure proof checking without revealing explicit data is required.

1.1 Problem Statement

Many different real-world scenarios rely on manual verification for both users and institutions. One example is where a prover $\mathcal{P}$ is going through a security screening when applying for a classified role, e.g. within the military or government sector. A prover $\mathcal{P}$ needs to prove a set of attributes such as no criminal record, no medical history of certain diseases/injuries etc. In a national database these attributes could be proven to exist or similarly, be proven as a non-membership in order to prove the lack of attributes. Other government agencies or medical institutions may have subsets of these attributes, hence are able to act as proxies when requesting the (non)-membership proofs for $\mathcal{P}$ to a verifer $\mathcal{V}$.

Intelligent Transportation Systems (ITS) and connected railway infrastructure are still on the rise, and with that many security implications as well [4, 5]. Personal devices (vehicle on-board devices, smartphones) and equipment such as cameras and radar sensory devices in the infrastructure can all be connected. A secure and privacy-preserving layer of data knowledge sharing is thus needed where a single device can prove attribute (data) knowledge or association efficient and through (multiple) proxies, e.g. a vehicle may need to prove eligibility for entering certain areas, or sensory devices must prove its geographical boundaries.

The given scenarios requires a multi-party setup where possibly a set of proxies $\mathcal{AP}$ is needed for proving that certain attributes $y_1, ..., y_n$ belong to $\mathcal{P}$, by proving that $y_1, ...y_n$ are securely stored in a trusted database at some database owner $\mathcal{S}$. Furthermore, proof of identity of all parties, including $\mathcal{P}$ and $\mathcal{S}$, and signatures for all proofs are needed to ensure authenticity and integrity. The assumption is that $\mathcal{P}$ either cannot provide such proofs, or prefer to delegate the proving part to one or many proxies, e.g. trusted parties such as a bank, hospital or in the IoT case intermediate servers. Therefore, we address the problem of delegated (non)-membership proving.

1.2 Accumulators and Proxy Signatures

A one-way *accumulator* is an efficient solution for secure set-membership proof problems, i.e. determine if a certain element belongs to a given set without revealing the elements. One important property of an accumulator is that it can efficiently provide a fixed-size *witness* for any element in the accumulator, which is used for verification of an element's set (non)-membership. The notion

of one-way accumulators was first proposed by Benaloh and de Mare [6]. Another type of cryptographic accumulators are constructions based on bilinear pairings and was first proposed by Nguyen [7].

Proxy signature protocols consists of three parties: *original signer*, *proxy signer* and *verifier*. A *warrant* is sent to the proxy, consisting of a predefined message space, a signature, identity of original signer etc. The warrant is used in combination with the proxy key to compute a secret proxy key for further signatures.

1.3 Related work

For the given use cases, tangent solutions exist addressing the privacy needs. In [8] it is demonstrated how a *private set intersection* (PSI) approach can be used in a cloud environment to calculate the set intersection on outsourced and encrypted data between client and server. However, the solution relies on encryption. Another approach is multi-party PSI solutions, first introduced in [9], where multiple data owners can prove data intersections among each other without revealing the non-intersecting parts. On the other hand, these and more recent protocols [10, 11] which also have the ability to make use of delegated private set intersection computations via proxies, still rely on encryption and does not use the proxies as intermediate verifiers. Hence, to our knowledge, there are no proposed protocols for using (multi) proxy signature schemes and accumulators as building blocks with merged key- and signature mechanisms in current literature.

1.4 Contribution

Our paper introduces the notion of *authenticated (multi) proxy accumulation* for solving both current and potentially new use cases. Our contribution consists of:

- Proposing a suitable explicit construction of combining a multi-proxy signature scheme with a dynamic accumulator scheme,
- proposing a general construction process for authenticated multi-proxy accumulation schemes,
- presents a security and correctness analysis for the proposed scheme,
- a performance analysis from our proof of concept implementation in Java and jPBC.

2 Preliminaries

A function $\mathcal{H} : \{0,1\}^* \rightarrow \mathbb{A}$ is a cryptographically secure hash function taking any binary string as input and produces an element of some set $\mathbb{A}$. Let $x_i \leftarrow_\$ X$ denote a randomized selection of x_i from a set X over a uniform distribution. A security parameter 1^λ determines parameter selection during initialization of a scheme, i.e. choosing suitable secure groups for a bilinear map, prime numbers and hash functions; thus λ corresponds to the provided n-bit security.

Definition 1 (Pairing). *Let $\mathbb{G}_1$, $\mathbb{G}_2$, $\mathbb{G}_T$ be groups with generators g_1, g_2, g_T respectively. Let $q = |\langle g_1 \rangle| = |\langle g_2 \rangle| = |\langle g_T \rangle|$. Let $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ be a bilinear map with the following properties:*

1. *Bilinearity:* $\forall (a, b \in \mathbb{Z}_q, g_1 \in \mathbb{G}_1, g_2 \in \mathbb{G}_2)$*:* $\hat{e}(g_1^a, g_2^b) = \hat{e}(g_1, g_2)^{ab}$,
2. *Computability: Computing $\hat{e}$ is efficient,*
3. *Non-degeneracy:* $\exists (g_1 \in \mathbb{G}_1, g_2 \in \mathbb{G}_2) : \hat{e}(g_1, g_2) \neq 1$.

we then say that e is a pairing *over groups $\mathbb{G}_1$, $\mathbb{G}_2$, $\mathbb{G}_T$.*

If $\mathbb{G}_1 = \mathbb{G}_2$, the pairing function is called *symmetric*, otherwise *asymmetric*.

Our construction use the Boneh-Lynn-Shacham (BLS) short signatures scheme [12]. BLS is provably secure under the Computational Diffie-Hellman problem and based on pairings. The signature of m is produced as $\sigma = \mathcal{H}_g(m)^{\mathtt{sk}}$ where `sk` is the signer's secret key. Verification is against the public key `pk` and the pairing $\hat{e}(\sigma, g) \stackrel{?}{=} \hat{e}(\mathcal{H}(m), \mathtt{pk})$.

3 General Scheme Construction Methodology

The benefit of using a modular construction as described here, is that each component can easily be changed when needed; if a scheme component is enhanced in the same security model the new component should seamlessly be interchanged. We propose a general approach, using a construction-by-modules principle, for constructing an authenticated multi-proxy accumulation scheme as follows:

1. **Scheme selection:** choose a signature scheme `Sig`, (multi)-proxy scheme `Proxy` and accumulator scheme `Acc` based on same cryptographic primitives (and possibly same hardness assumption), e.g. bilinear pairings.
2. **Hardness selection:** Make sure the hardness assumptions are compatible, e.g. for pairings, check that the underlying pairing schemes are all symmetric or asymmetric, to avoid mismatches during the security analysis.
3. **Re-usage:** Make sure `Sig` can be re-used for all steps in the `Proxy` protocol and does not rely on several different signature schemes.
4. **Extension:** Extend the `Proxy` protocol to use one additional round of signature generation/verifying, as described in Section 4.2 for the *request* and *proving* phases, i.e. the proxies first compute an intermediate round of signature checking with the set owner, and then the final signature round to the verifying party.
5. **Security Analysis:** Make sure the security of the merged parts of the scheme can be reduced to the security assumption of the `Sig` component.

In conclusion, the merge of `Sig`, `Proxy` and `Acc` builds on the efficiency of using the same cryptographical primitives and the flexibility to easily add a second round of signature checking with the data owner to perform an intermediate proof checking step. Naturally, this scales linearly; for n proxies only $2n$ additional signature checks are needed.

4 Authenticated Multi-Proxy Accumulation Scheme

4.1 System model

The model consists of four parties: set owner $\mathcal{S}$, accumulation proxy $\mathcal{AP}$, prover $\mathcal{P}$ and verifier $\mathcal{V}$. Set owner $\mathcal{S}$ has full control over a finite set $Y = \{y_1, ..., y_n\}$ for which prover $\mathcal{P}$ wants to prove membership of some element $y_i \in Y$ for verifer $\mathcal{V}$. In this particular setting, $\mathcal{P}$ must delegate the proving part to $\mathcal{AP}$ which in turn communicates the proof to verifier $\mathcal{V}$. This implies $\mathcal{V}$ to verify three things: the validity of the membership proof, that $\mathcal{P}$ is authenticated and thus implicitly the delegation of the proof from $\mathcal{P}$ to $\mathcal{AP}$ also follows.

4.2 Proposed Construction

We propose an *authenticated multi-proxy accumulation scheme* (AMPA). It involves a multi-party setup with a prover $\mathcal{P}$, a set of (at least one) accumulation proxies $\mathcal{AP} = \{\mathcal{AP}_1, \mathcal{AP}_2, ..., \mathcal{AP}_n\}$, a set authority $\mathcal{S}$ and a verifier $\mathcal{V}$. An element $y_j \in Y$ is considered an attribute of $\mathcal{P}$ and stored securely. Moreover, y_j is committed to the set authority's database, i.e $y_j \in S$ where set S is securely stored and handled by $\mathcal{S}$. Our scheme consists of 7 algorithms:

$\mathsf{Setup}(1^\lambda, n) \rightarrow (\mathsf{par}, pk_S, sk_S, acc_\emptyset, state_\emptyset)$**:** generates a tuple of bilinear pairing parameters and necessary secure hash functions $\mathsf{par} = (q, \mathbb{G}_1, \mathbb{G}_2, g, \hat{e}, \mathcal{H}_1, \mathcal{H}_2)$ to publish, where $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_2$ is a bilinear map and $\mathcal{H}_1, \mathcal{H}_2$ are collision-resistant hash functions such that $\mathcal{H}_1 : \{0,1\}^* \rightarrow \mathbb{Z}_q^*$ and $\mathcal{H}_2 : \{0,1\}^* \rightarrow \mathbb{G}_1$. Next, signature key-pair (sk, pk) for set authority $\mathcal{S}$, is generated as $sk = x_0 \xleftarrow{\$} \mathbb{Z}_q^*$ and $pk = g^{x_0}$. Accumulator key-pair (sk_S, pk_S) for $\mathcal{S}$ is generated as key tuples $sk_S = (\gamma \xleftarrow{\$} \mathbb{Z}_q^*, sk)$ and $pk_S = (pk, \hat{e}(g,g)^{\gamma^{n+1}})$. Additionally, the accumulator $acc_\emptyset = 1$ with state table $state_\emptyset$ initiated. Finally, the prover $\mathcal{P}$ generates a keypair as $sk_P = x_1 \xleftarrow{\$} \mathbb{Z}_q^*$ and $pk_P = g^{x_1}$.

$\mathsf{KeyExtract}(\{1, 2, ..., \mathsf{l}\}) \rightarrow L$**:** generates set of key-pairs for proxy signers $L = \{(pk_1, sk_1), (pk_2, sk_2), ..., (pk_l, sk_l)\}$. Either a suitable key agreement protocol can be used if there is a single node running this procedure, otherwise each proxy signer runs this procedure locally and broadcast the public key to all other parties.

$\mathsf{ProxyKeyGen}(w, S_{O_w}) \rightarrow sk_{AP_i}$**:** given a warrant w and S_{O_w} issued and generated by original signer, the proxy $\mathcal{AP}_i$ invokes $\mathsf{ProxyKeyGen}$ and verifies that $e(S_{O_w}, g) == e(\mathcal{H}_2(w), pk_P)$. If valid then the proxy signing key is computed as $sk_{AP_i} = S_{O_w} + \mathcal{H}_2(w)^{sk_i}$. Note that $S_{O_w} = \mathcal{H}_2(w)^{x_0}$.

$\mathsf{ProxyAccSignWitness}(m_1, w) \rightarrow \sigma_1$**:** The proxy generates $k_{AP_i} \xleftarrow{\$} \mathbb{Z}_p^*$ and computes $r_{AP_i} = \hat{e}(g,g)^{k_{AP_i}}$. Each r_{AP_i} is broadcast to all other $\mathcal{AP}_i$'s who computes $r_{AP} = \Pi_{i=1}^{l} r_{AP_i}$, $c_{AP} = \mathcal{H}_1(m_1 || r_{AP})$ and $U_{AP_i} = (sk_{AP_i})^{c_{AP}} + g^{k_{AP_i}}$. One designated $\mathcal{AP}_i$, called the *clerk* verifies $r_{AP_i} = \hat{e}(U_{AP_i}, g)(\hat{e}(\mathcal{H}_2(w), pk + pk_i))^{-c_{AP}}$ for $i = 1, 2, .., l$, and if successful

computes $U_{AP} = \Sigma_{i=1}^{l} U_{AP_i}$ and sends signature $\sigma_1 = (m_1, c_{AP}, U_{AP}, w)$ to $\mathcal{S}$. Message $m_1 = (\omega_1, acc_P, g||y_j)$ where ω_1 is the witness of y_j from $\mathcal{P}$, $acc_{\mathcal{P}}$ the accumulator of $\mathcal{P}$ and $g||y_j$ which is used during accumulation verification as in [13].

ProxyAccVerifyWitness$(\sigma_1) \rightarrow \sigma_2$**:** $\mathcal{S}$ receives the witness and element for verification from $\mathcal{AP}$. Next, $\mathcal{S}$ verifies that $y_j \in Y$ using the witness and check that $c_{AP} = \mathcal{H}_1(m_1||r_{AP})$; note that $r_{AP} = \hat{e}(U_{AP}, g)(\hat{e}(\mathcal{H}_2(w), pk + \Sigma_{i=1}^{l} pk_i))^{-c_{AP}}$. If so, $\mathcal{S}$ signs $m_2 = (\omega_{i_{AP}}, acc_P, \omega_{i_S}, acc_S)$ implying that $y_j \in S$. Signature procedure is same as in ProxyAccSignWitness but with $\mathcal{S}$ as only signer (hence clerk), and returns $\sigma_2 = (m_2, c_{\mathcal{S}}, U_{\mathcal{S}}, w)$.

ProxyAccSignProof$(\sigma_2) \rightarrow \sigma_3$**:** All proxies runs ProxyAccSignWitness over σ_2 as message. Resulting signature is $\sigma_3 = (\sigma_2, c_{\mathcal{AP}}, U_{\mathcal{AP}}, w)$.

ProxyAccVerifyProof$(\sigma_3) \rightarrow \{\perp, true\}$**:** $\mathcal{V}$ verifies σ_3 by computing $c_{\mathcal{AP}} = \mathcal{H}_1(\sigma_2||r_{\mathcal{AP}})$. If correct then parse σ_2 and if needed checks membership of $y_j \in S$. Note that if the signature verifies correctly, $\mathcal{V}$ implicitly knows that $y_j \in Y, S$, that $\mathcal{AP}$ is a designated signer and $\mathcal{S}$ is a valid set authority.

Note that four more algorithms, AccAdd, AccUpdate, AccWitUpdate and AccVerify, are used just as they are stated in [13]. The complete protocol executes in phases described below: *setup, request* and *proving phases*. Note that we assume $\mathcal{S}$ is running a trusted environment.

Setup phase : $\mathcal{P}$ securely sends and commit set Y to set authority $\mathcal{S}$ who updates the complete set S such that $Y \subseteq S$. $\mathcal{S}$ then runs Setup and publishes all public parameters such as groups and pairing function, accumulator value $\mathtt{acc}_{\mathcal{S}}$ and generates associated signature- and accumulator keys $sk_{\mathcal{S}}, pk_{\mathcal{S}}$. All proxies runs KeyExtract to get their own key-pairs (these has to be exchanged using a secure key exchange protocol). Finally, $\mathcal{P}$ sends a warrant w to the collection of proxy signers $\mathcal{AP}_1, ..., \mathcal{AP}_l$ who then runs ProxyKeyGen to generate specific proxy signature keys sk_{AP_i} associated to $\mathcal{P}$ and $\mathcal{S}$.

Request phase : Verifier $\mathcal{V}$ asks $\mathcal{P}$ to prove membership of $y_j \in S$. This triggers $\mathcal{P}$ to send $w, \omega_1, acc_P, g||y_j$, i.e warrant, witness for $y_i \in Y$, accumulator value and the element concatenation needed for the membership proof, to all relevant proxies $\mathcal{AP}$ who in turn create a signature σ_1 over the tuple by invoking ProxyAccSignWitness. Next step is that $\mathcal{AP}$ sends σ_1 to $\mathcal{S}$ who responds with witness ω_2 if the signature can be verified using ProxyAccVerifyWitness and that accumulation membership proof of $y_j \in Y$ holds.

Proving phase : $\mathcal{AP}$ runs ProxyAccSignProof to generate a final proof σ_3 which is a signature over σ_2 and contains information such as membership proof (witness), delegation proof and authenticity proof of $\mathcal{P}$,$\mathcal{AP}$ and $\mathcal{S}$. Verifer $\mathcal{V}$ runs ProxyAccVerifyProof function that uses AccVerify as subroutine, to verify membership proof $y_j \in S$. $\mathcal{V}$ responds either *true* or error symbol $\perp$ to $\mathcal{P}$.

Accumulators $acc_{\mathcal{P}}$ and $acc_{\mathcal{S}}$ ($\mathcal{P}$'s and $\mathcal{S}$'s accumulators respectively) and their associated states $state_{\mathcal{P}}, state_{\mathcal{S}}$, are variables we do not consider in the scheme definition.

5 Security Analysis

5.1 Proof of Correctness

We note that ProxyAccVerifyWitness verifies correctly and computes signature σ_2 if and only if $y \in Y$ and $c_{AP} = \mathcal{H}_1(m_1||r_{AP})$ holds. Since $\mathcal{S}$ verifies $y \in Y$ accordingly to [13], then if successful, ProxyAccVerifyWitness procedure verifies $c_{AP} = \mathcal{H}_1(m_1||r_{AP})$. This is only possible if r_{AP} is correctly generated. Moreover, ProxyAccVerifyProof verifies σ_3 correctly if and only if $y \in Y$ and c_{AP} holds. The proof for that follows since we set $l = 1$ for the number of proxies, thereby only using one proxy instead of a full collection.

Theorem 1. ProxyAccVerifyWitness *verifies correctly and computes signature* σ_2 *if and only if* $y \in Y$ *and* $c_{AP} = \mathcal{H}_1(m_1||r_{AP})$ *holds.*

Proof. We note that $\sigma_1 = (m_1, c_{AP}, U_{AP}, w)$, thus c_{AP} is parsed. Also, $pk = g^{x_0}$ and $pk_i = g^{x_i}$. We denote $\mathcal{H}_2(w) = h_2$ for readability. Next, since

$$= \hat{e}(U_{AP}, g) \left(\hat{e}(h_2, pk + \sum_{i=1}^{l} pk_i) \right)^{-c_{AP}} \tag{1}$$

$$= \hat{e}(\sum_{i=1}^{l} (sk_{AP_i})^{c_{AP}} + g^{k_{AP_i}}, g) \left(\hat{e}(h_2, g^{x_0}) \prod_{i=1}^{l} \hat{e}(h_2, g^{x_i}) \right)^{-c_{AP}} \tag{2}$$

$$= \left(\prod_{i=1}^{l} \hat{e}(sk_{AP_i}, g)^{c_{AP}} \hat{e}(g, g)^{k_{AP_i}} \right) \left(\hat{e}(h_2, g^{x_0}) \prod_{i=1}^{l} \hat{e}(h_2, g^{x_i}) \right)^{-c_{AP}} \tag{3}$$

$$= \left(\prod_{i=1}^{l} \hat{e}(h_2^{x_0}, g)^{c_{AP}} \hat{e}(h_2^{x_i}, g)^{c_{AP}} \hat{e}(g, g)^{k_{AP_i}} \right) \left(\hat{e}(h_2, g^{x_0}) \prod_{i=1}^{l} \hat{e}(h_2, g^{x_i}) \right)^{-c_{AP}} \tag{4}$$

$$= \prod_{i=1}^{l} \hat{e}(g, g)^{k_{AP_i}} = r_{AP} \tag{5}$$

we can verify that $c_{AP} = \mathcal{H}_1(m_1||r_{AP})$. $\mathcal{S}$ verifies $y \in Y$ accordingly to [13], thus $\mathcal{S}$ can successfully generate σ_2 using same procedure as for σ_1 but with $m_2 = (y_i, \omega_{i_{AP}}, acc_P, g||y_i, \omega_{i_S}, acc_S)$. □

Theorem 2. ProxyAccVerifyProof *verifies correctly and computes signature* σ_3 *if and only if* $y \in Y$ *and* $c_{AP} = \mathcal{H}_1(m_2||r_{AP})$ *holds.*

Proof. Same as in Theorem 1 but for $l = 1$. □

5.2 Security Model and Analysis

We consider a security experiment where forger $\mathcal{F}$ is allowed to query signatures σ_1, σ_2 and σ_3 given public parameters. Let $\mathcal{O}_{\mathsf{Sign}}(\alpha, m, i) \rightarrow \sigma_i$ be a signing

oracle which returns a valid signature $\sigma_i, i \in \{1,2,3\}$, m a message and $\alpha = \{\mathsf{par}, pk_1, pk_2, ...\}$ is the set of public parameters and all public keys necessary. Moreover, let $\mathcal{O}_{\mathsf{H}_1}(x) \rightarrow c$ be a random oracle which return elements $c \in \mathbb{Z}_q^*$, given some binary string x, thus emulating $\mathcal{H}_1$, and similarly $\mathcal{O}_{\mathsf{H}_2}(x) \rightarrow d$ where $d \in \mathbb{G}$.

Definition 2 (Security Experiment). *Let π be an AMPA scheme initialized with $\mathsf{Setup}(1^\lambda, n)$ and all proxy signature keys generated. Next, let $\mathcal{F}$ be a polynomial-time forgery algorithm with the ability to query $\mathcal{O}_{\mathsf{Sign}}$, $\mathcal{O}_{\mathsf{H}_1}$ and $\mathcal{O}_{\mathsf{H}_2}$ a polynomial number of times (q times). After a maximum of q queries, $\mathcal{F}$ is able to generate a signature tuple $(\sigma_1^*, \sigma_2^*, \sigma_3^*)$ for messages m_1^*, m_2^* that has not been previously queried. We then say that π is secure if*

$$Pr[\mathcal{F}^{\mathcal{O}_{\mathsf{Sign}}, \mathcal{O}_{\mathsf{H}_1}, \mathcal{O}_{\mathsf{H}_2}}(\alpha) \rightarrow (\sigma_1^*, \sigma_2^*, \sigma_3^*, m_1^*, m_2^*) \wedge \mathsf{Verify}(\sigma_1^*, \sigma_2^*, \sigma_3^*) = 1] < \epsilon \quad (6)$$

where Verify *returns 1 if and only if subroutines* $\mathsf{ProxyAccVerifyWitness}(\sigma_1^*) = \sigma_2^*$, $\mathsf{ProxyAccSignProof}(\sigma_2^*) = \sigma_3^*$ *and* $\mathsf{ProxyAccVerifyProof}(\sigma_3^*) = 1$, *and ϵ is negligible.*

Theorem 3 (Non-forgeability). *The proposed AMPA scheme is secure against forgery as defined in Definition 2.*

Proof (Sketch of proof). The security experiment initializes according to Def. 2 and $\mathbb{G}$ is a computationally secure Diffie-Hellman group. We consider two cases: (1) where forger $\mathcal{F}$ compute σ_3^* directly and (2) when $\mathcal{F}$ compute σ_2^* and use it for further computations to achieve σ_3^*. We omit a third case where σ_1^* is computed since the proof is same as case (1) since σ_1 and σ_3 only differs over which message to sign, i.e. either m_1 or σ_2.

Case 1: Assume forger $\mathcal{F}$ manages to compute a forged signature σ_3^*. We note that a signature $\sigma_3^* = (\sigma_2^*, c_{\mathcal{AP}}^*, U_{\mathcal{AP}}^*, w)$ where $\sigma_2^* = (m_2^*, c_{\mathcal{S}}^*, U_{\mathcal{S}}^*, w)$. In such forgery we get that $\mathcal{O}_{\mathsf{H}}(m_2) \rightarrow c'$ thus

$$U_{\mathcal{AP}}^* = \sum_{i=1}^{l} \left((sk_{AP_i})^{c'} + g^{k_{AP_i}} \right) = \sum_{i=1}^{l} \left((S_{O_w} + \mathcal{H}_2(w)^{sk_i})^{c'} + g^{k_{AP_i}} \right) \quad (7)$$

$$= \sum_{i=1}^{l} \left((\mathcal{H}_2(w)^{x_0} + \mathcal{H}_2(w)^{x_i})^{c'} + g^{k_{AP_i}} \right) \quad (8)$$

Since σ_3^* is a forgery and validates correctly by $\mathsf{ProxyAccVerifyProof}$, the signing oracle needs to be a BLS-oracle $\mathcal{O}_{\mathsf{BLS}}$ (using $\mathcal{O}_{\mathsf{H}_2}$ as a subroutine), i.e. returning valid BLS signatures $\mathcal{H}_2(w)^{x_0}$ and $\mathcal{H}_2(w)^{x_i}$ after at most q queries. Therefore, c' will not help $\mathcal{F}$ in breaking the scheme and

$$Pr[\mathcal{F}^{\mathcal{O}_{\mathsf{Sign}}, \mathcal{O}_{\mathsf{H}_1}, \mathcal{O}_{\mathsf{H}_2}}(\alpha)] \leq Pr[\mathcal{F}^{\mathcal{O}_{\mathsf{BLS}}}(\alpha)] \leq \epsilon \quad (9)$$

since BLS is provably secure in the random oracle model with a reduction to breaking the computational Diffie-Hellman problem [12].

Case 2: Assuming $\mathcal{F}$ manages to compute a forged signature σ_2^*, then similar to case (1) the forged signature contains $U_{\mathcal{S}}^* = (sk_S)^{c'} + g^{k_S} = \left((\mathcal{H}_2(w)^{x_0} + \mathcal{H}_2(w)^{x_s})^{c'} + g^{k_S}\right)$ where x_s is the secret key of $\mathcal{S}$ and $k_S \in \mathbb{Z}_p^*$ chosen randomly by $\mathcal{S}$. Again, the forgery implies a BLS-oracle, hence the scheme is secure. □

6 Implementation

The Java Pairing-Based Cryptography library (jPBC) is a Java port of the PBC library written in C which provides the mathematical operations needed for pairing-based cryptosystems [14]. Computations were over a field of 318-bit modulo. In order to better understand the efficiency of our protocol, a set of different operations and procedures went through a performance analysis, measuring the approximate time in milliseconds. Each operation and procedure was executed 1000 times on a MacBook Pro, 2017, with 2.3 GHz Dual-Core Intel Core i5, 16 GB 2133 MHz LPDDR3 on macOS Big Sur 11.2. The results are presented in Table 1. As expected, all procedures containing pairings were the slowest. We strongly expect our results to be much faster using a more efficient implementation of the hash-to-group algorithm. The scalability analysis consider running the ProxyAccSignWitness procedure with different number of proxies. It seems to scale linearly as expected, and for 1000 participating proxies the proxy signature- and verification procedure takes roughly 1 minute.

Operation	**Time**(ms)	**Ops.**
$\mathbb{G}$: point addition	0.0003	4
$\mathbb{G}$: point multiplication	0.0006	3
$\mathbb{Z}$: exponentiation	0.0310	10
Hash to $\mathbb{G}$	21.5461	5
Pairing	5.5111	7
Proxies	**Time** (ms)	
10	728.72	
100	6591.15	
1000	63847.88	

Procedure	**Time**(ms)
ProxyAccSetup	31.04
ProxyAccKeyExtract	9.76
ProxyAccProxyKeyGen	51.18
ProxyAccSignWitness	59.83
ProxyAccVerifyWitness	55.97
ProxyAccSignProof	59.57
ProxyAccVerifyProof	54.46
Total run	316.14

Table 1: Performance Analysis. Ops is number of operations.

7 Conclusion

We provided a method for constructing AMPA schemes, illustrating how to combine them into a practical protocol along with a proof-of-concept implementation and performance analysis. We have also shown the validity

of the intact security analysis covering the merge of two schemes, showing the correctness and non-forgeability. We conclude that important aspects to consider in our merge methodology is to choose an overlapping underlying hardness assumption for the combined schemes, utilizing the same key-pairs and reuse signature procedures as a second layer between proxies and database owner(s).

References

1. Colleen Graham, Neha Gupta, Vanitha Dsilva, Michael Warrilow, Brandon Medford, and Chad Eschinger. Forecast: Public cloud services, worldwide, 2018-2024, 3q20 update, 2020.
2. Marcin Rojszczak. Cloud act agreements from an eu perspective. *Computer Law & Security Review*, 38:105442, 09 2020.
3. Netwrix. 2021 cloud data security report. [Online; accessed 20-June-2021].
4. Dalton A Hahn, Arslan Munir, and Vahid Behzadan. Security and privacy issues in intelligent transportation systems: Classification and challenges. *IEEE Intell. Transp. Syst. Mag.*, 13(1):181–196, 2021.
5. Ayyoub Lamssaggad, Nabil Benamar, Abdelhakim Senhaji Hafid, and Mounira Msahli. A survey on the current security landscape of intelligent transportation systems. *IEEE Access*, 9:9180–9208, 2021.
6. Josh Benaloh, Michael de Mare, and Giordano Automation. One-Way Accumulators: A Decentralized Alternative To Digital Signatures. pages 274–285. Springer-Verlag, 1993.
7. Lan Nguyen. Accumulators from bilinear pairings and applications. In *Topics in Cryptology – CT-RSA 2005*, pages 275–292. Springer Berlin Heidelberg, 2005.
8. Q. Zheng and S. Xu. Verifiable delegated set intersection operations on outsourced encrypted data. In *2015 IEEE International Conference on Cloud Engineering*, pages 175–184, 2015.
9. Michael J. Freedman, Kobbi Nissim, and Benny Pinkas. Efficient private matching and set intersection. In Christian Cachin and Jan L. Camenisch, editors, *Advances in Cryptology - EUROCRYPT 2004*, pages 1–19, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.
10. Atsuko Miyaji and Shohei Nishida. A scalable multiparty private set intersection. In *Network and System Security*, pages 376–385, Cham, 11 2015. Springer International Publishing.
11. Aydin Abadi, Sotirios Terzis, and Changyu Dong. Vd-psi: Verifiable delegated private set intersection on outsourced private datasets. In Jens Grossklags and Bart Preneel, editors, *Financial Cryptography and Data Security*, pages 149–168, Berlin, Heidelberg, 2017. Springer Berlin Heidelberg.
12. Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the weil pairing. In *Advances in Cryptology–ASIACRYPT '01, LNCS*, pages 514–532. Springer, 2001.
13. Jan Camenisch, Markulf Kohlweiss, and Claudio Soriente. An accumulator based on bilinear maps and efficient revocation for anonymous credentials. In *Public Key Cryptography – PKC 2009*, pages 481–500. Springer Berlin Heidelberg, 2009.
14. Angelo De Caro and Vincenzo Iovino. jpbc: Java pairing based cryptography. In *Proceedings of the 16th IEEE Symposium on Computers and Communications, ISCC 2011*, pages 850–855. IEEE, 2011.