An Authenticated Accumulator Scheme for Secure Master Key Access in Microservice Architectures

Hannes Salin[1] and Dennis Fokin[2]
[1]*Department of Information and Communication Technology, Swedish Transport Administration, Borlänge, Sweden*
[2]*Independent Researcher, Sweden*

Keywords: Key Management, Microservices, Cryptographic Accumulators, Pairing-based Cryptography, Secure Key Storage.

Abstract: We consider the use-case of Internet of Things ecosystems with an API-driven microservice architecture, where the need for accessing cryptographic functions is crucial. Devices communicate with a microservice backend, which in turn integrates to a secure key storage in order to compute digital signatures or message encryption. Access to a secure storage must be executed securely and naive approaches such as passwords or certificates are used in practice today, which still may be open to impersonation attacks. With the usage of efficient cryptographic accumulators we therefore propose a secure key access scheme with the microservices ecosystem in mind, and provide initial results from a proof-of-concept implementation in Java and jPBC, where we show a performance- and communication complexity analysis. Finally, we provide a security analysis of the scheme in the random oracle model.

1 INTRODUCTION

Standardization of Internet of Things (IoT) Representational State Transfer (REST) architecture is still ongoing (Keränen et al., 2021) and addresses what is nowadays a well-known type of architecture in software engineering. Using REST API:s is well aligned with an underlying microservice architecture - it is far from unusual to adopt a microservices architecture over a monolithic one; with higher demands of scalability and resilience, it also gives the benefit of seamlessly combining different modules of a system, developed with different technology stacks. An API-driven pattern enables this modularity, but also gives rise to several security related challenges. Authentication and authorization seems to be the most commonly addressed security mechanisms in a microservices context (Pereira-Vale et al., 2019). However, despite a range of industry best practices on how to approach these challenges, not much is scientifically established for the general case of authenticating services for secure access. Given an ecosystem of IoT devices built on a microservices architecture, where possibly a subset of these need frequent access to cryptographic functions such as digital signatures or encryption, requirements for strong authentication and secret key access are needed. One approach is to store secret keys on each and every service. However, that would lead to severe security implications since an adversary would only have to compromise the service in order to gain access to the secret key. In some cases (e.g. for transaction signatures), a master key is needed, thus a single point of storage must exist. This problem is often addressed by using Hardware Security Modules (HSM) and/or vaults, for secure storage. On the other hand, to access an HSM or vault, a password, key or similar is still needed for authenticated access; this problem seems unsolvable since regardless of how far we apply a layer of security, an entity still needs to authenticate itself at some point.

1.1 Problem Statement

In an IoT-driven environment with clusters of devices connecting to backend services, i.e. a microservices-based ecosystem, with provided functionality that include cryptographic operations (such as digital signatures or encrypting data), the secret key(s) must be securely stored. We consider particularly high-sensitive systems in connected environments such as Intelligent Transport Systems (ITS) where both vehicles and infrastructure can be connected. With many types of IoT devices such as cameras, radars, and other sensory devices, data needs to be transmitted to interme-

Salin, H. and Fokin, D.
An Authenticated Accumulator Scheme for Secure Master Key Access in Microservice Architectures.
DOI: 10.5220/0010989400003194
In *Proceedings of the 7th International Conference on Internet of Things, Big Data and Security (IoTBDS 2022)*, pages 119-126
ISBN: 978-989-758-564-7; ISSN: 2184-4976

diate nodes; different types of traffic data needs to be exchanged between providers and/or consuming parts of the backend services (NordicWay, 2021). Also, the European Commission has initiatives considering the inclusion of end user smartphone data collection and distribution for traffic planning purposes, using API-driven solutions (Joinup, 2021).

We need to investigate a suitable solution for key access procedures in a HSM, from a microservice node. The access must have enough security but still balance between performance and scalability. Typical REST API:s can have layers of security for the actual API calls, but we also need to ensure that the protocol that triggers an HSM operation is authenticated and sound; TLS is a good way to secure the communication, but the request procedure of accessing the secret key/crypto operation within the HSM must also not be breakable or leak sensitive data that can compromise a client or HSM.

1.2 Cryptographic Accumulators

A *cryptographic accumulator* is an efficient algorithm for proving set-membership. Given an element y in some set Y, it is possible for a *prover* to prove for a *verifier* that $y \in Y$ without revealing any other information of the elements. By using a (fixed-size) *witness* for an element in the accumulator, a verification of that element's set (non)-membership is possible. The security lies in the hardness of computing such witness for elements not belonging to the set Y, i.e. forge a membership proof. The notion of one-way accumulators was first proposed by Benaloh and de Mare (Benaloh et al., 1993), known as RSA accumulators. To illustrate the idea, we give a short overview of a simple RSA accumulator function $f(x) = A^x \bmod N$ where A is the accumulator (holding all the elements) and $N = pq$ for primes p, q. Initially the accumulator consists of the element $A_0 = g$, then after accumulating x_1 we get $A_1 = g^{x_1}$, and for a sequence of elements $x_2, x_3, \ldots, x_n$ we get $A_n = (g^{x_1})^{x_2 x_3, \ldots x_n} = g^{x_1 x_2 x_3 \ldots x_n}$. Now, to create a witness for element x_i we compute $w_i = g^{x_1 \cdots x_{i-1} x_{i+1} \cdots x_n}$, and to prove that $x_i \in A_n$ we can simply verify that $w_i^{x_i} = g^{x_1 \ldots x_{i-1} x_i x_{i+1} \ldots x_n} = A_n$.

Cryptographic accumulators can be categorized into *static, dynamic* and *universal* types. Static accumulators cannot alter the set, i.e adding/removing elements, whereas dynamic accumulators have this ability. Universal accumulators provide both alteration and the possibility to perform verification of non-membership proofs, i.e. prove that $y \notin Y$.

1.3 Related Work

A recent survey indicates that both products and programming libraries are available for most types of key access solutions, e.g. software vaults and cryptographic primitives for Secure Multi-Party Computations (SMPC) (Salin and Fokin, 2021). Regarding SMPC and similar solutions, nothing has become widely adopted as of yet in the industry, although seemingly on the rise. Moreover, identified naive approaches for key access are storing the secret key embedded in the application (during build), in a database with stored database credentials in a property file or storing pointers to the actual key in a configuration file. These solutions imply direct vulnerability if the device or node is compromised.

A typical approach in industry is using a traditional public key infrastructure (PKI) for microservice authentication, hence the usage of certificates between nodes. In (Pahl and Donini, 2018) a certificate-based methodology for IoT authentication is presented, where additional authentication data is embedded in each executable. The proposal shows promising scalability and could most likely be used for a microservice authenticated access to a key storage. However, the solution still builds on PKI key management which handles revocation and key distribution in a non-efficient way in a large-scale deployment. In some scenarios, authentication could be a collaborative approach, i.e. a set of microservices that would need to split some access credential among them and perform a SMPC scheme. Harn (Harn, 2012) proposed a group authentication protocol designed specifically to be a many-to-many authentication system, instead of the more classical approaches of having a one-to-one relationship, e.g. one prover and one verifier. This was accomplished by using Shamir's secret sharing. We could view this in a microservice perspective where the group is a set of IoT backend services, computing access collaboratively. While this solution is efficient, an adversary could gain access to one of the shares and through a replay attack become authenticated. A further attack vector with Shamir's secret sharing scheme is that a party could become compromised and send a non-legitimate secret to the entity collecting the secrets. This would break the secret sharing protocol. Another secret sharing approach for key management has been proposed (Li et al., 2014) with the angle of adding a layer of data security by including derived convergent keys for data encryption, secured by a master key. However, as noted previously, secret sharing schemes also come with certain drawbacks, and is therefore on its own not a complete solution for the situation researched in this paper.

1.4 Contribution

We propose a novel solution to securely provide master key access in a microservice ecosystem, constructing an *accumulator-based* scheme with *witness splitting* and an interactive request access mechanism. Our contribution consists of:

- a proposed secure master key access scheme with witness splitting and a dynamic cryptographic accumulator,
- a security and correctness analysis of the proposed scheme,
- a proof-of-concept implementation of the scheme in Java with a performance- and communication complexity analysis.

2 PRELIMINARIES

We introduce a set of cryptographic primitives necessary for building our proposed scheme. The accumulator is based on a dynamic non-adaptive scheme in (Karantaidou and Baldimtsi, 2021), which in turn is proven secure under the q-Strong Diffie-Hellman Assumption (q-SDH). A secure hash function $\mathcal{H}$ is a collision-free function $\mathcal{H} : \{1,0\}^* \to \mathbb{G}$ resistant to preimage attacks. We denote $x \xleftarrow{\$} \mathbb{A}$ to be an element x chosen randomly (uniformly) from a set $\mathbb{A}$.

Definition 1. *Let $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ be multiplicative groups of prime order p. A* pairing *is a bilinear map $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ with the following properties:*

- ***Bilinearity**: For all $x \in \mathbb{G}_1, y \in \mathbb{G}_2$ then $\hat{e}(x^a, y^b) = \hat{e}(x,y)^{ab}$ for any $a, b \in \mathbb{Z}$.*
- ***Non-degeneracy**: For generators $g_1 \in \mathbb{G}_1, g_2 \in \mathbb{G}_2$ then $\hat{e}(g_1, g_2) \neq 1$.*
- ***Efficient**: The map $\hat{e}$ can be efficiently computed.*

If $\mathbb{G}_1 = \mathbb{G}_2$ then we call the pairing *symmetric*.

Definition 2. *Let $k \in \mathbb{N}$ be a security parameter and $\mathbb{G}$ a cyclic group of order $q > 2^k$. Let g be a generator of $\mathbb{G}$. The* Discrete Logarithm Problem *(DLP) is, given a randomly chosen $y, g \in \mathbb{G}$, to find the unique $x \in \mathbb{Z}_q^*$ such that $y = g^x$.*

Definition 3 (q-Strong Diffie-Hellman Assumption). *Let g be a randomly chosen generator to the cyclic group $\mathbb{G}$, with prime order p, and a randomly chosen element $x \in \mathbb{Z}_p$. Given the set $\{g^{x^i} | i \in \mathbb{Z}_q \cup \{0\}\}$ for some integer q, a probabilistic polynomial-time (PPT) adversary can only compute the pair $(c, g^{\frac{1}{x+c}}) \in \mathbb{Z}_p \times \mathbb{G}$ with a negligible probability.*

The Boneh-Lynn-Shacham (BLS) short signature scheme (Boneh et al., 2001) is used in our proposed architecture for creating specific client witness values. BLS is based on pairings and consists of four procedures where signatures are computed by $\sigma = \mathcal{H}(m)^{\mathtt{sk}}$ with secret key $\mathtt{sk}$. A signature is then validated by $\hat{e}(\sigma, g) \stackrel{?}{=} \hat{e}(\mathcal{H}(m), \mathtt{pk})$ where $\mathtt{pk}$ is the public key of the signer.

2.1 System Settings

Let $\mathsf{ID}_1, ..., \mathsf{ID}_n$ be a set of n microservices; we refer to each node as a *client*. Each service is running on a trusted, hardened (secured) container in an orchestration platform, handling the communication pipeline between all nodes. Another node, $\mathcal{HSM}$, is a secure storage service containing a master key k and can perform cryptographic functions $\mathbf{f}_k(\cdot)$.

We assume an IoT device can trigger implicit access to the $\mathcal{HSM}$ node by requesting a service at some ID_i, either from an IoT interface integrating directly with ID_i or as part of the microservice ecosystem. Communication is via API:s between the nodes, assumed securely established using TLS or similar well-known deployable solutions. The authentication of the IoT device in the public network, is abstracted on the surrounding system-level, e.g. by tokenization/federation or other type of authentication framework, terminating before ID_i requests access to $\mathcal{HSM}$, thus out-of scope and not relevant for our paper.

3 ACCUMULATOR MASTER KEY ACCESS SCHEME

Our proposed master key access scheme builds on a similar accumulator construction given in (Karantaidou and Baldimtsi, 2021), but with an interactive additive witness update mechanism based on the discrete logarithm problem. We use the accumulator to securely keep track of eligible clients that should have access, and use the interactive witness mechanism to ensure forward secrecy and authentication. Clearly, similar types of accumulator constructions based on pairings can be used as a cryptographic primitive; we illustrate the idea with the dynamic non-adaptive scheme in (Karantaidou and Baldimtsi, 2021). By using an accumulator, a single value can be stored in the HSM instead of a database with identities, key values and so on. This approach would reduce the complexity of key management since revocation is simply equivalent to removing the key access value for a client in the accumulator. Another strength in our approach is also the ease of implementation and the increased difficulty for an adversary to simply steal a password or secret key from a client; since each access attempt generates an ephemeral intermediate value, combined with a witness splitting technique, no replay attacks are possible.

3.1 Overview

A master key k is stored in a secure module $\mathcal{HSM}$. A client with identity ID needs access to compute a cryptographic function $\mathbf{f}_k(m)$, e.g. a signature or encryption over a message m. In order to compute $\mathbf{f}_k(m)$ and thereby implicitly accessing k, following procedure will establish an authenticated access:

- ID publish public key $\mathsf{pk}_{\mathsf{ID}}$.
- ID and $\mathcal{HSM}$ agrees on security parameters $\mathsf{par} = \{\mathbb{G}, q, g, \hat{e}, \mathcal{H}\}$.
- ID computes registry value $\sigma_{\mathsf{ID}} = \mathcal{H}(ID)^{\mathsf{sk}_{\mathsf{ID}}}$ and sends to $\mathcal{HSM}$.
- $\mathcal{HSM}$ derives a client-specific *access value* $k_{\mathsf{ID}} = \mathcal{H}(ID)^{\mathsf{sk}_{\mathsf{ID}}} + \mathcal{H}(ID)^{\mathsf{sk}_{\mathcal{HSM}}}$.
- $\mathcal{HSM}$ accumulates k_{ID} into a client access accumulator and generates a witness $w_{\mathsf{ID}} = \mathsf{GenWitness}(k_{\mathsf{ID}})$.
- $\mathcal{HSM}$ splits w_{ID} into w_{ID_1} and w_{ID_2} using SplitWitness and immediately updates the shares using the UpdateWitness procedure.
- $\mathcal{HSM}$ share w_{ID_1} to ID.
- At this stage, $\mathcal{HSM}$ is securely storing k_{ID} in the accumulator and the client possess the partial witness w_{ID_1}, thus the client is not storing the complete witness nor the access value (or even has any knowledge of what that value is).
- When the client wants access, it sends w_{ID_1} to $\mathcal{HSM}$.
- $\mathcal{HSM}$ responds with a challenge g^{z_i} where $z_i \xrightarrow{\$} \mathbb{Z}_q^*$ for the ith iteration of request.
- The client computes $(g^{z_i})^{\mathsf{sk}_{\mathsf{ID}}} = g^{z_i \cdot \mathsf{sk}_{\mathsf{ID}}}$ and sends to $\mathcal{HSM}$.
- $\mathcal{HSM}$ verifies that $\hat{e}(g^{z_i \cdot \mathsf{sk}_{\mathsf{ID}}}, g^{-z_i}) = \hat{e}(g,g)^{\mathsf{sk}_{\mathsf{ID}}} = \hat{e}(g, \mathsf{pk}_{\mathsf{ID}})$, since the inverse to z_i is only known to $\mathcal{HSM}$.
- If successful both parties updates $w_{C_1'} = w_{C_1} + g^{z_i \cdot \mathsf{sk}_{\mathsf{ID}}}$, and $\mathcal{HSM}$ also updates $w_{\mathsf{ID}_2'} = (w_{\mathsf{ID}_2})^{z_i} - g^{z_i \cdot \mathsf{sk}_{\mathsf{ID}}}$, using the UpdateWitness procedure, thus invalidates the old shares.

In the last step, when updating the partial witness, we refer to a mechanism where the witness w_{ID} can be split into $w_{\mathsf{ID}_1}, w_{\mathsf{ID}_2}$, and later in time gets updated in such a way that the new shares are bound to an ephemeral value and the secret key of the client. Relying on the discrete logarithm problem, the challenge-response part with $g^{z_i \cdot \mathsf{sk}_{\mathsf{ID}}}$ ensure that $\mathcal{HSM}$ can verify the authenticity of ID and thereby proceed with the key updating procedure. We illustrate the complete sequence of our scheme in Tab. 1.

3.2 Scheme Description

We define our master key access scheme as follows:

$\mathsf{Setup}(1^\lambda) \to (\mathsf{par})$**:** generates secure Diffie-Hellman groups, pairing function and hash functions, collected into $\mathsf{par} = \{\mathbb{G}, q, g, \hat{e}, \mathcal{H}\}$. To initialize an empty accumulator, a random value $u \xleftarrow{\$} \mathbb{Z}_q^*$ is generated and the starting accumulator value is set to $v_0 = g^u$. The $\mathcal{HSM}$ itself also generates a private signature key $\mathsf{sk}_{\mathcal{HSM}}$. The client also generates public and secret keys $\mathsf{pk}_{\mathsf{ID}}, \mathsf{sk}_{\mathsf{ID}}$.

$\mathsf{Register}(\sigma_{\mathsf{ID}}, \mathsf{ID}) \to (\perp, w_{ID_1})$ **:** A client with identity ID register access to the $\mathcal{HSM}$ using a signed proof $\sigma_{\mathsf{ID}} = \mathcal{H}(ID)^{\mathsf{sk}_{\mathsf{ID}}}$ and receives the partial witness w_{ID_1} after the $\mathcal{HSM}$ has verified against $\mathsf{pk}_{\mathsf{ID}}$. Register function is computed as follows:

$$k_{\mathsf{ID}} = \sigma_{\mathsf{ID}} + \mathcal{H}(ID)^{\mathsf{sk}_{\mathcal{HSM}}} \tag{1}$$

$$= \mathcal{H}(\mathsf{ID})^{\mathsf{sk}_{\mathsf{ID}}} + \mathcal{H}(\mathsf{ID})^{\mathsf{sk}_{\mathcal{HSM}}} \tag{2}$$

Next, procedures Accumulate and SplitWitness are used as subroutines in order to finally return w_{ID_1} to the client.

$\mathsf{Accumulate}(k_{\mathsf{ID}}) \to w_{\mathsf{ID}}$**:** accumulates k_{ID} and generates a witness w_{ID} for the access key of ID (eligible for access). If k_{ID} is not already inserted, the accumulation from state t to $t+1$ is computed as $v_{t+1} = v_t^{k_{\mathsf{ID}} + \mathsf{sk}_{\mathcal{HSM}}}$ and the witness $w_{\mathsf{ID}} = (v_{t+1})^{\frac{1}{k_{\mathsf{ID}} + \mathsf{sk}_{\mathcal{HSM}}}}$.

$\mathsf{SplitWitness}(w_{\mathsf{ID}}) \to w_{\mathsf{ID}_1}, w_{\mathsf{ID}_2}$**:** splits the witness into two shares, using the following formula where $s \xleftarrow{\$} \mathbb{G}_1$:

$$w_{\mathsf{ID}_1} = s \tag{3}$$

$$w_{\mathsf{ID}_2} = w_{\mathsf{ID}} - s \tag{4}$$

$\mathsf{UpdateWitness}(w_{\mathsf{ID}_1}, w_{\mathsf{ID}_2}, g^{z \cdot \mathsf{sk}_{\mathsf{ID}}}) \to w_{\mathsf{ID}_{1'}}, w_{\mathsf{ID}_{2'}}$**:** updates the witness shares using $g^{z \cdot \mathsf{sk}_{\mathsf{ID}}}$ where $z \xleftarrow{\$} \mathbb{Z}_q^*$:

$$w_{\mathsf{ID}_1'} = w_{\mathsf{ID}_1} + g^{z \cdot \mathsf{sk}_{\mathsf{ID}}} \tag{5}$$

$$w_{\mathsf{ID}_2'} = (w_{\mathsf{ID}_2})^z - g^{z \cdot \mathsf{sk}_{\mathsf{ID}}} \tag{6}$$

Table 1: Sequence diagram of the accumulator-based secure master key access scheme.

Client ID	Message	$\mathcal{HSM}$
$\mathsf{sk_{ID}} \xleftarrow{\$} \mathbb{Z}_q^*$, $\mathsf{pk_{ID}} = g^{\mathsf{sk_{ID}}}$	$\mathsf{par} = \{\mathbb{G}, q, g, \hat{e}, \mathcal{H}, v_0\}$	$k \xleftarrow{\$} \mathbb{Z}_q^*, \mathsf{sk}_{\mathcal{HSM}} \xleftarrow{\$} \mathbb{Z}_q^*$
$\sigma_{\mathsf{ID}} = \mathcal{H}(\mathsf{ID})^{\mathsf{sk_{ID}}}$	$\sigma_{\mathsf{ID}} \xrightarrow{\qquad}$	
		$k_{\mathsf{ID}} = \sigma_{\mathsf{ID}} + \mathcal{H}(\mathsf{ID})^{\mathsf{sk}_{\mathcal{HSM}}}$
		$w_{\mathsf{ID}}, v_t = \mathsf{Accumulate}(k_{\mathsf{ID}})$
	$\xleftarrow{\qquad} w_{\mathsf{ID}_1}$	$(w_{\mathsf{ID}_1}, w_{\mathsf{ID}_2}) = \mathsf{SplitWitness}(w_{\mathsf{ID}})$
Request access to $\mathbf{f}_k(\cdot)$	$\sigma_{\mathsf{ID}}, w_{\mathsf{ID}_1} \xrightarrow{\qquad}$	
	$\xleftarrow{\qquad} g^z$	$z \xleftarrow{\$} \mathbb{Z}_q^*$
$(g^z)^{\mathsf{sk_{ID}}}$	$g^{z\cdot\mathsf{sk_{ID}}}, m \xrightarrow{\qquad}$	
		$\hat{e}(g^{-z}, g^{z\cdot\mathsf{sk_{ID}}}) \overset{?}{=} \hat{e}(g, \mathsf{pk_{ID}})$
		$w_{\mathsf{ID}} = \mathsf{CombineWitness}(w_{\mathsf{ID}_1}, w_{\mathsf{ID}_2})$
		$\mathbf{f}_k(m) = \mathsf{Access}(w_{\mathsf{ID}}, \sigma_{\mathsf{ID}}, m)$
Updates and store $w_{\mathsf{ID}'_1}$		$(w_{\mathsf{ID}'_1}, w_{\mathsf{ID}'_2}) = \mathsf{UpdateWitness}(w_{\mathsf{ID}_1}, w_{\mathsf{ID}_2}, g^{z\cdot\mathsf{sk_{ID}}})$

after successful verification of $\hat{e}(g^{-z}, g^{z\cdot\mathsf{sk_{ID}}}) \overset{?}{=} \hat{e}(g, \mathsf{pk_{ID}})$.

$\mathsf{CombineWitness}(w_{\mathsf{ID}'_1}, w_{\mathsf{ID}'_2}) \to w_{\mathsf{ID}}$**:** combines the witness shares as follows:

$$w_{\mathsf{ID}'_1} + \left(w_{\mathsf{ID}'_2} + g^{z\cdot\mathsf{sk_{ID}}}\right)^{\frac{1}{z}} \tag{7}$$

$$= w_{\mathsf{ID}_1} + g^{z\cdot\mathsf{sk}_{\mathcal{ID}}} + ((w_{\mathsf{ID}_2})^z)^{\frac{1}{z}} \tag{8}$$

$$= w_{\mathsf{ID}_1} + w_{\mathsf{ID}_2} = w_{\mathsf{ID}}. \tag{9}$$

$\mathsf{Access}(w_{\mathsf{ID}}, \sigma_{\mathsf{ID}}, m) \to \mathbf{f_k}(m)$**:** A client accessing function $\mathbf{f_k}(\cdot)$ which uses the secret master key k to perform a cryptographic operation. Uses the witness and verifies that k_{ID} is accumulated. Witness checking for k_{ID} is done by verifying $\hat{e}(w_{\mathsf{ID}}, g^{k_{\mathsf{ID}}} g^{\mathsf{sk}_{\mathcal{HSM}}}) = \hat{e}(v_{t+1}, g)$.

4 SECURITY ANALYSIS

4.1 Security Model and Setup

The underlying accumulator scheme is shown to be secure under the q-SDH assumption in the appendix of (Karantaidou and Baldimtsi, 2021). However, since we use it as a cryptographic building block we provide an overall security analysis of our scheme and focus on those parts surrounding the accumulation verification and unforgeability.

We build our security analysis on the *random oracle model* (ROM), using a set of oracles that a probabilistic polynomial-time (PPT) adversary $\mathcal{A}$ can utilize. The Combine procedure in the scheme is modeled as an oracle that can be queried a polynomial q number of times, serving the adversary fresh values.

Definition 4. *Let O_{Combine} be a* combine oracle *which takes a witness share w_{ID_j} and outputs a valid witness w^*_{ID}, i.e. $O_{\mathsf{Combine}}(w_{\mathsf{ID}_j}) = w^*_{\mathsf{ID}}$ such that $\mathsf{Access}(w^*_{\mathsf{ID}}, \sigma^*, m) = 1$ for any message m and corresponding signature σ^*.*

We also consider a hash (signature) oracle, similar to O_{Combine}, but instead returns a signature $\sigma^* = \mathcal{H}(x)^{\mathsf{sk_{ID}}}$:

Definition 5. *Let $O_{\mathsf{Sign,ID}}$ be a* signature oracle *which returns a BLS signature on any binary input string* ID *using the secret key of* ID*. The signature will validate successfully, i.e. $\mathsf{Access}(w^*_{\mathsf{ID}}, \sigma^*, m) = 1$ given a valid witness w^*_{ID}.*

Definition 6. *Let* sub *be a polynomial-time subroutine which extracts a secret key, given σ^* and a witness.* sub *returns the secret key of either the client or $\mathcal{HSM}$.*

Definition 7 (Security Experiment I). *We define the security experiment $\mathsf{EXP}_{\mathcal{A}}^{O_{\mathsf{Sign,ID}}}$ as follows: the adversary $\mathcal{A}$ receives a valid witness w^*_{ID} to use once (i.e. no updates of shares will leak to the adversary) for requesting access. The adversary can only extract one key during the experiment using* sub*. $\mathcal{A}$ wins the game if the advantage*

$$\mathcal{ADV}_{\mathcal{A}}(\mathsf{EXP}_{\mathcal{A}}^{O_{\mathsf{Sign,ID}}}) = Pr[(O_{\mathsf{Sign,ID}}(\mathsf{ID}) = \sigma^*_{\mathsf{ID}}) \wedge \left(\mathsf{sub}(\sigma^*_{\mathsf{ID}}, w^*_{\mathsf{ID}}) = \mathsf{sk_{ID}} \vee \mathsf{sk}_{\mathcal{HSM}}\right)] \tag{10}$$

is not negligible, i.e. the probability of success for $\mathcal{A}$ to extract one of the secret keys in order to forge k_{ID} and thereby be able to access $\mathcal{HSM}$ in any forthcoming requests.

Definition 8 (Security Experiment II). *We define the security experiment $\mathsf{EXP}_{\mathcal{A}}^{O_{\mathsf{Combine}}}$ as follows: the* *adversary $\mathcal{A}$ receives a previous share $w_{\mathsf{ID}_{j,i-1}}$ where $j \in \{1,2\}$ and can query O_{Combine} for a combined witness q times, except for the latest updated witness containing z_i. $\mathcal{A}$ is assumed to have access to the client signature $\sigma = \mathcal{H}(\mathsf{ID})^{\mathsf{sk}_{\mathsf{ID}}}$. $\mathcal{A}$ wins the game if the advantage*

$$\mathcal{ADV}_{\mathcal{A}}(\mathsf{EXP}_{\mathcal{A}}^{O_{\mathsf{Combine}}}) = Pr[(O_{\mathsf{Combine}}(w_{\mathsf{ID}_{1,i-1}}) = w^*_{\mathsf{ID}}) \wedge (\mathsf{Access}(w^*_{\mathsf{ID}}, \sigma, m) = 1)] \quad (11)$$

is not negligible, i.e. the probability of success for $\mathcal{A}$ generating a valid witness given an old witness share. This would demonstrate the security against replay attacks with regards to witness shares.

4.2 Analysis

We need to validate that an accumulated value k_{ID} is correctly verified by the witness w_{ID}, and that the subroutine using Access is correctly executing $\mathbf{f}_k(\cdot)$ when the client is authenticated, and rejects otherwise.

Theorem 1. *Algorithms* Accumulate *and* Access *verifies correctly.*

Proof. We note that Accumulate implicitly validates by returning the witness for the accumulated value, e.g. k_{ID} will generate w_{ID}, bound to ID and $\mathcal{HSM}$ in state $t+1$:

$$w_{\mathsf{ID}} = (v_{t+1})^{\frac{1}{k_{\mathsf{ID}}+\mathsf{sk}_{\mathcal{HSM}}}} = (v_t)^{\frac{k_{\mathsf{ID}}+\mathsf{sk}_{\mathcal{HSM}}}{k_{\mathsf{ID}}+\mathsf{sk}_{\mathcal{HSM}}}} = v_t \quad (12)$$

Thus the witness correctly represents the existence of k_{ID} in the accumulator. Next, we consider the correctness of the proof checking mechanism:

$$\hat{e}(w_{\mathsf{ID}}, g^{k_{\mathsf{ID}}} g^{\mathsf{sk}_{\mathcal{HSM}}}) = \hat{e}(v_t, g^{k_{\mathsf{ID}}} g^{\mathsf{sk}_{\mathcal{HSM}}}) \quad (13)$$
$$= \hat{e}(v_t, g)^{k_{\mathsf{ID}}+\mathsf{sk}_{\mathcal{HSM}}} \quad (14)$$
$$= \hat{e}(v_t^{k_{\mathsf{ID}}+\mathsf{sk}_{\mathcal{HSM}}}, g) \quad (15)$$
$$= \hat{e}(v_{t+1}, g) \quad (16)$$

□

Theorem 2. *The $\mathcal{HSM}$ node verifies the ephemeral value $g^{z\cdot\mathsf{sk}_{\mathsf{ID}}}$ correctly.*

Proof. In the intermediate step of the protocol we need to ensure that the $\mathcal{HSM}$ node correctly verifies the ephemeral value before proceeding with the witness combination procedure. The check consists of the pairing $\hat{e}(g^{-z}, g^{z\cdot\mathsf{sk}_{\mathsf{ID}}}) \overset{?}{=} \hat{e}(g, \mathsf{pk}_{\mathsf{ID}})$ and we prove its correctness (note that $-z$ is the inverse element to z, i.e. $z \cdot (-z) = (-z) \cdot z = 1$):

$$\hat{e}(g^{-z}, g^{z\cdot\mathsf{sk}_{\mathsf{ID}}}) = \hat{e}(g,g)^{(-z)\cdot z\cdot\mathsf{sk}_{\mathsf{ID}}} \quad (17)$$
$$= \hat{e}(g,g)^{\mathsf{sk}_{\mathsf{ID}}} \quad (18)$$
$$= \hat{e}(g, \mathsf{pk}_{\mathsf{ID}}) \quad (19)$$

□

Theorem 3. *Procedures* Register *and* Access *preserves the un-forgeability of signatures in the scheme, i.e. secure under the security experiment of type I.*

Proof. The client signature is given by a BLS signature $\sigma_{\mathsf{ID}} = \mathcal{H}(\mathsf{ID})^{\mathsf{sk}_{\mathsf{ID}}}$, and further the client access value is only an addition of the $\mathcal{HSM}$'s version of signature, resulting in $k_{\mathsf{ID}} = \mathcal{H}(\mathsf{ID})^{\mathsf{sk}_{\mathsf{ID}}} + \mathcal{H}(\mathsf{ID})^{\mathsf{sk}_{\mathcal{HSM}}}$. Knowledge of either term in k_{ID} will not give any advantage in finding the secret keys due to the security of BLS (Boneh et al., 2001). Therefore the probability

$$Pr[(O_{\mathsf{Sign},\mathsf{ID}}(\mathsf{ID}) = \sigma^*_{\mathsf{ID}}) \wedge (\mathsf{sub}(\sigma^*_{\mathsf{ID}}) = \mathsf{sk}_{\mathsf{ID}})] \quad (20)$$

is negligible since sub must then be an algorithm breaking BLS. Also, knowing both terms and being able to extract k_{ID} will not break the scheme in any way since the final verification is independent of verifying k_{ID} itself, but rather verifies that k_{ID} is correctly accumulated.

Now, let $\mathcal{A}$ be an adversary having access to a signing oracle $O_{\mathsf{Sign},\mathsf{ID}}$ and a witness $w^*_{\mathsf{ID}} = (v_{t+1})^{\frac{1}{k_{\mathsf{ID}}+\mathsf{sk}_{\mathcal{HSM}}}}$. It is shown in the appendix of (Karantaidou and Baldimtsi, 2021) that accumulation is secure under the q-SDH assumption, hence w^*_{ID} will not give the adversary any advantage in extracting the secret keys since both are again secured as BLS signatures and now also in the exponent of the accumulation, thus

$$Pr[\mathsf{sub}(w^*_{\mathsf{ID}}) = (\mathsf{sk}_{\mathsf{ID}} \vee \mathsf{sk}_{\mathcal{HSM}})] \quad (21)$$

is negligible. Finally, above result implies that $\mathcal{ADV}_{\mathcal{A}}(\mathsf{EXP}_{\mathcal{A}}^{O_{\mathsf{Sign},\mathsf{ID}}})$ is negligible. □

Our scheme uses a split-and-refresh technique for the witness, often used for lightweight private key management (Selvi et al., 2019; Ferreira and Dahab, 2002). The idea is to split the secret value into two shares and after every usage, a random value is generated and used for recomputing the shares in a way such that the combined value is still the original one. Afterwards the previous shares are deleted from memory.

Theorem 4. *Algorithms* SplitWitness *and* CombineWitness *correctly split and updates w_{ID}, invalidating any previous, outdated share w_{ID_i}, i.e. secure under the security experiment of type II.*

Proof. Consider witness w_{ID}, first splitted in the initial stage by SplitWitness and directly updates, thus the first time the two shares are $w_{\mathsf{ID}'_1} = w_{\mathsf{ID}_1} + g^{z_1\cdot\mathsf{sk}_{\mathsf{ID}}}$ and $w_{\mathsf{ID}'_2} = (w_{\mathsf{ID}_2})^{z_1} - g^{z_1\cdot\mathsf{sk}_{\mathsf{ID}}}$. Clearly the witness is combined efficiently as $w_{\mathsf{ID}_1} + g^{z_1\cdot\mathsf{sk}_{\mathsf{ID}}} + (w_{\mathsf{ID}_2})^{z_1\cdot\frac{1}{z_1}} -$

$g^{z_1 \cdot \mathsf{sk}_{\mathsf{ID}}} = w_{\mathsf{ID}}$. Next, for the ith consecutive iteration of updating the key shares, we get

$$s + \sum_{i=1}^{n} + g^{z_i \cdot \mathsf{sk}_{\mathsf{ID}}} - s - \sum_{i=1}^{n} g^{z_i \cdot \mathsf{sk}_{\mathsf{ID}}} + w_{\mathsf{ID}} = w_{\mathsf{ID}}. \quad (22)$$

Assume the ith iteration of $\mathcal{HSM}$ access occur and denote the current key shares as $w_{\mathsf{ID}_{1,i}}$ and $w_{\mathsf{ID}_{2,i}}$. Next, assume an attacker get access to a previous key share $w_{\mathsf{ID}_{1,i-1}}$. This implies that the attacker also lacks the latest g^{z_i} since the previous share is computed on $g^{z_{i-1}}$, or explicitly $w_{\mathsf{ID}_1,i-1} = w_{\mathsf{ID}_1,i-2} + g^{z_{i-1}\mathsf{sk}_{\mathsf{ID}}}$. Since $\mathsf{CombineWitness}$ combines correctly, the attacker needs either $\mathsf{sk}_{\mathsf{ID}}$ or a value α such that $\mathsf{CombineWitness}(\alpha, w_{\mathsf{ID}_2,i-1}) = w_{\mathsf{ID}}$. For the first case, the attacker need to break the DLP since $\mathsf{sk}_{\mathsf{ID}}$ is only previously used in computations in the exponent of $\mathcal{H}(\mathsf{ID})^{\mathsf{sk}_{\mathsf{ID}}}$ and $g^{z_{i-1}\cdot \mathsf{sk}_{\mathsf{ID}}}$. In the second case it must hold that (assuming the attacker can force $\mathcal{HSM}$ to execute $\mathsf{CombineWitness}$ with arbitrarily z's, i.e. using oracle O_{Combine}):

$$\alpha + (w_{\mathsf{ID}_2,i-1})^{\frac{z_{i-1}}{z_{i-1}}} - g^{z_{i-1}\cdot \mathsf{sk}_{\mathsf{ID}}} \quad (23)$$

$$= \alpha + w_{\mathsf{ID}_2,i-1} - g^{z_{i-1}\cdot \mathsf{sk}_{\mathsf{ID}}} \quad (24)$$

$$= w_{\mathsf{ID}} \quad (25)$$

which implies $\alpha = w_{\mathsf{ID}} + g^{z_{i-1}\cdot \mathsf{sk}_{\mathsf{ID}}}$ thus the attacker need both witness shares, or again need to solve the DLP (to extract $\mathsf{sk}_{\mathsf{ID}}$) in order to solve α. The same argument holds if the attacker have access to a previous share $w_{\mathsf{ID}_1,i-1}$ and we conclude that

$$Pr[O_{\mathsf{Combine}}(\alpha) = w^*_{\mathsf{ID}} \land \mathsf{Access}(w^*_{\mathsf{ID}}, \sigma, m) = 1] \quad (26)$$

is negligible due to the hardness of DLP, thus we conclude that the scheme is secure against replay attacks of the witness shares. $\square$

5 IMPLEMENTATION

Our proof-of-concept implementation includes a scalable microservices ecosystem, built on a containerization architecture to emulate a typical real-world implementation as closely as possible. Each container contains a service with a REST API for communication. The implementation is written in Java and the Spring Boot framework, together with the Java Pairing-Based Cryptography library (jPBC). This is a Java porting of a C library called PBC and provides the mathematical framework needed for pairing-based cryptography (De Caro and Iovino, 2011). For our implementation, the Type A pairings were chosen for its simplicity. These are constructed on the curve $y^2 = x^3 + x$ over a field $\mathbb{F}_q$ for some prime $q = 3 \mod 4$. Group $\mathbb{G}_1$ is of prime order r where r is also a prime factor of $q+1$. Our chosen q was a 318-bit integer. Note that operations are computed as standard group element operations on the elliptic curve, i.e. curve point addition and multiplication.

5.1 Cryptographic Execution Efficiency

The protocol consists of six functions, noted in Tab. 3. In order to attain a clear understanding of how the scheme affects efficiency, every function went through a performance analysis measuring the approximate time in milliseconds. Each function was executed 100 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. An analysis of each separate mathematical operation was also measured, given in Tab 2.

Table 2: Performance analysis of mathematical operations where e are elements (points) on the pre-configured elliptic curve, and n are 318-bit integers.

Operation	Time (*ms*)
e.pow(n)	17.59
$\mathsf{e_i}$.powZn($\mathsf{e_j}$)	8.98
$\mathsf{e_i}$.add($\mathsf{e_j}$)	0.04
pairing	4.85
e.mul(n)	17.45
$\mathsf{e_i}$.mul($\mathsf{e_j}$)	0.04
hash(n)	0.019
random(e)	2.78
random(n)	0.051

Table 3: Performance analysis of the scheme's procedures.

Function	Time (*ms*)
setup()	36.79
register()	0.31
accumulate()	26.79
splitWitness()	2.37
updateWitness()	63.17
combineWitness()	104.64
access()	75.38

5.2 Communication Efficiency

We conducted a brief communication analysis in terms of efficiency of sending and receiving requests/responses between a microservice node and the $\mathcal{HSM}$ node. Using TLS for secure communication is industry standard; however it will give a significant overhead in communication complexity due to TLS handshake procedures and other types of establishments for new connections. We measured our implementation (one microservice) repeatedly with TLS and without TLS, in order to understand the connection overhead. For 1000 repeated new connections the mean proportion of using TLS compared to not using TLS was $\Delta = 0.9227$, i.e. about 92% of the total connection time is dedicated to the TLS protocol.

Table 4: Communication analysis of the scheme's API connections.

API	no TLS (*ms*)	TLS (*ms*)
/registerClient	96.657	540.98
/requestAccess	169.18	926.81
/proveAccess	175.54	985.12
Total execution time:	441.38	2452.91

The API consists of three main endpoints: /registerClient is the initial registration of a client including key access value generation, /requestAccess is for requesting access and thus handle the ephemeral value $g^{z \cdot \mathsf{sk}_{ID}}$, and finally /proveAccess will trigger the actual crypto function $\mathbf{f}_k(\cdot)$. These HTTP endpoints take a set of parameters in base64 string format and are converted into byte streams when handled by the node. To conclude the experiment, the timing results for our proof of concept implementation lies within reasonable timings. As a comparison, a microservice node in our experiment that only connects to a secure website and receives a HTTP OK response, had a mean timing of 768.95 ms.

6 CONCLUSION

We have shown a theoretical construction of an authenticated accumulator-based master key access scheme, proven its security under type I and II security experiments, i.e. secure against forgery and reply attacks. Our proof-of-concept implementation shows promising results indicating the feasibility and easy-to-adopt approach in a microservices context. We argue that the level of implementation is significantly easier using wrapper crypto libraries such as jPBC, than developing code using low-level languages; it is also closer to real-world microservices applications where high-level programming languages such as Java and C# is de facto choice in industry.

REFERENCES

Benaloh, J., de Mare, M., and Automation, G. (1993). One-Way Accumulators: A Decentralized Alternative To Digital Signatures. pages 274–285. Springer-Verlag.

Boneh, D., Lynn, B., and Shacham, H. (2001). Short signatures from the weil pairing. In *Advances in Cryptology–ASIACRYPT '01, LNCS*, pages 514–532. Springer.

De Caro, A. and Iovino, V. (2011). jpbc: Java pairing based cryptography. In *Proceedings of the 16th IEEE Symposium on Computers and Communications, ISCC 2011*, pages 850–855. IEEE. [Online; accessed 08-February-2022].

Ferreira, L. C. and Dahab, R. (2002). Blinded-key signatures: securing private keys embedded in mobile agents. In *In Proceedings of the 2002 ACM symposium on Applied computing (ACM SAC'02*, pages 82–86. ACM Press.

Harn, L. (2012). Group authentication. *IEEE Transactions on computers*, 62(9):1893–1898.

Joinup (2021). Intelligent transport systems - cooperative, connected and automated mobility (its-ccam) and electromobility (rp2020). https://joinup.ec.europa.eu/collection/rolling-plan-ict-standardisation/. [Online; accessed 27-November-2021].

Karantaidou, I. and Baldimtsi, F. (2021). Efficient constructions of pairing based accumulators. In *2021 2021 IEEE 34th Computer Security Foundations Symposium (CSF)*, pages 373–388, Los Alamitos, CA, USA. IEEE Computer Society.

Keränen, A., Kovatsch, F. M., and Hartke, K. (2021). Guidance on restful design for internet of things systems. Internet-Draft draft-irtf-t2trg-rest-iot-08, IETF Secretariat. https://www.ietf.org/archive/id/draft-irtf-t2trg-rest-iot-08.txt.

Li, J., Chen, X., Li, M., Li, J., Lee, P. P., and Lou, W. (2014). Secure deduplication with efficient and reliable convergent key management. *IEEE Transactions on Parallel and Distributed Systems*, 25(6):1615–1625.

NordicWay (2021). Interchange node under the nordic way project. https://github.com/NordicWayInterchange. [Online; accessed 08-February-2022].

Pahl, M.-O. and Donini, L. (2018). Securing iot microservices with certificates. In *NOMS 2018 - 2018 IEEE/IFIP Network Operations and Management Symposium*, pages 1–5.

Pereira-Vale, A., Márquez, G., Astudillo, H., and Fernandez, E. B. (2019). Security mechanisms used in microservices-based systems: A systematic mapping. In *2019 XLV Latin American Computing Conference (CLEI)*, pages 01–10.

Salin, H. and Fokin, D. (2021). Mission impossible: Securing master keys.

Selvi, S. S. D., Paul, A., Rangan, C. P., Dirisala, S., and Basu, S. (2019). Splitting and aggregating signatures in cryptocurrency protocols. In *2019 IEEE International Conference on Decentralized Applications and Infrastructures (DAPPCON)*, pages 100–108.