2023 IEEE 29th International Conference on Parallel and Distributed Systems (ICPADS) | 979-8-3503-3071-7/23/$31.00 ©2023 IEEE | DOI: 10.1109/ICPADS60453.2023.00242

SLCSA: Scalable Layered Cooperative Service Attestation Scheme in Cloud-Edge-End Cooperation Environments

Jie Cui
School of Computer Science and Technology
Anhui University
Anhui, China
cuijie@mail.ustc.edu.cn

Qipeng Chen
School of Computer Science and Technology
Anhui University
Anhui, China
e21201030@stu.ahu.edu.cn

Li Han*
School of Computer Science and Technology
Anhui University
Anhui, China
hanli98@ahu.edu.cn

Yang Li
Anhui Province Key Laboratory of Cyberspace Security Situation Awareness and Evaluation
Anhui, China
liyanghf@nudt.edu.cn

Qingyang Zhang
School of Computer Science and Technology
Anhui University
Anhui, China
qingyang.zhang.inchina@gmail.com

Lu Liu
School of Informatics
University of Leicester
Leicester, U.K
l.liu@leicester.ac.uk

Hong Zhong
School of Computer Science and Technology
Anhui University
Anhui, China
zhongh@ahu.edu.cn

***Abstract*—In a cloud-edge-end cooperation environment, edge and core cloud services are complementary and synergistic, jointly processing a large amount of private data uploaded by users. To prevent the leakage of private data, users must ensure that services are secure and trusted through remote attestation. Traditional one-to-one remote attestation schemes are typically used to test the cloud services. However, as the cloud platform scales and the number of edge and core cloud services grows rapidly, the traditional attestation method has problems, such as poor scalability and low attestation efficiency. Thus far, there has been a lack of feasible methods for users to verify multiple related services in a cloud-edge-end cooperation environment quickly. This paper presents a scalable layered cooperative service attestation (SLCSA) scheme, the first secure and scalable protocol for the efficient attestation of multiple cooperative services. The SLCSA scheme is based on a Boneh–Lynn–Shacham (BLS) multisignature to improve the scalability of the scheme while enabling users to conduct the batch verification of services. We also analyze the security of the proposed scheme. To evaluate the proposed scheme, we implement it using Intel SGX, which can provide basic hardware-assisted attestation and a trusted execution environment for services. The experimental results show that the SLCSA scheme is practical and efficient in a cloud-edge-end cooperative environment.**

***Index Terms*—collective remote attestation, cloud-edge-end cooperation environments, multisignature, Intel SGX, Trusted Execution Environment (TEE)**

I. Introduction

In recent years, the rapid development of technologies, such as the Internet of things (IoT) [1] and cloud computing [2], has brought profound changes to the Internet industry. With the massive growth in connected devices, there is a high demand for cloud-computing models. Edge computing has gained increasing attention in the past few years as a new computing model for cloud-computing environments [3]. Compared with traditional cloud computing, a hybrid solution that includes an edge cloud can significantly relieve network bandwidth and the computational pressure on cloud servers. It accelerates data processing and improves service responsiveness to support IoT applications. The edge cloud service is primarily responsible for aggregating partial data in the domain, completing the analysis and reasoning of the sensory data, and transmitting relevant analysis results to the end devices [4]. However, the massive number of end devices at the edge side of the network continuously generate a large amount of sensory data and send these data to the edge cloud for processing, which may result in insufficient resources [5]. At this point, the resources of the core cloud service can be called to supplement those required to satisfy the requirements of edge-side applications. This is called "cloud-edge-end cooperation." Edge and core cloud services that together provide computing resources to users are called cooperative services. In this environment, end devices may have high mobility, which severely affects application reliability and service quality [6]. In addition, because of the task offloading strategy and serving handover, the edge and core cloud services used by the end device are not fixed [7]. Consequently, end devices may use multiple services provided by the edge and core clouds in a short period.

In this case, it is critical for users to ensure the reliability of the multiple cooperative services; that is, that no malicious code is running on the cloud. Currently, the attestation of cloud services is usually performed using two traditional one-to-one

*Li Han is the corresponding author.

2690-5965/23/$31.00 ©2023 IEEE
DOI 10.1109/ICPADS60453.2023.00242

remote attestation methods: the Private Certificate Authority (PCA) and Direct Anonymous Attestation (DAA) schemes [8], [9]. To make the service transition smooth and improve the user experience and efficiency of the cloud platform, multiple cooperative services need to be verified simultaneously. However, it is not possible to verify several cooperative services in a short period using traditional attestation methods.

Collective remote attestation technology provides a new direction for addressing the challenges of multiservice attestation. This technology can verify a number of devices in a relatively short time. Thus far, most collective remote attestation technologies have focused on the low-end group of devices in the IoT, and these schemes typically require a static network topology to be maintained throughout the attestation process. For example, in the SANA collective remote attestation protocol [10], attestation results are aggregated by intermediate nodes. A parent node must receive the attestation results from all its children before it can continue to send attestation replies to the upper layers. However, in a cloud-edge-end cooperation environment, the network topology changes dynamically owing to task offloading and resource-allocation strategies. To date, there is a lack of viable methods for users to verify multiple cooperative services with dynamic topologies quickly. This study aimed to leverage the idea of collective remote attestation to design a multiservice attestation scheme suitable for a cloud-edge-end cooperation environment that can improve the verification efficiency of users for edge and core cloud services while ensuring system security.

In this paper, we present a collective remote attestation scheme for a cloud-edge-end cooperation environment. The solution is based on a Boneh-Lynn-Shacham (BLS) multisignature [11] and modifies SANA [10] for the cloud-edge-end cooperation environment. This enables users to verify the reliability of multiple cooperative services quickly. We call our scheme "scalable layered cooperative service attestation (SLCSA)." The contributions of this study are as follows.

- To the best of our knowledge, the SLCSA scheme is the first collective remote attestation protocol proposed for cloud-edge-end cooperation environments. The SLCSA scheme can not only quickly and effectively verify all the cooperative services used by the users of a cloud platform, but also detect malicious cooperative services to safeguard user privacy and system security. Moreover, it does not require the network topology to remain static during the attestation process.
- The SLCSA uses multisignatures to improve the verification efficiency. The cooperative service acts as a prover to generate a public–private key pair and sends it to the cloud platform administrator for registration.
- We implement the SLCSA protocol using the Intel SGX technology [12]. The running time of each phase of the protocol is also evaluated. In addition, we analyze the security of the protocol. This proves that the proposed solution can be integrated into large cloud platforms as a basic service. When users need to verify a large number of cooperative services simultaneously, they can invoke our protocol to verify multiple services quickly.

The remainder of this paper is organized as follows. In Section II, we discuss related works on traditional one-to-one remote attestation and collective remote attestation. We introduce the system assumptions and adversary model in Section III. Section IV presents the preliminaries and notations. Section V describes the SLCSA protocol. Subsequently, we present the performance results of the SLCSA scheme in Section VI. A security analysis of SLCSA is conducted in Section VII. Finally, we conclude the paper in Section VIII.

II. Related Work

Traditional remote attestation can be divided into three major types: software-based, hardware-based, and hybrid. Software-based attestation [13] does not require the device to have secure hardware. However, these schemes are based on assumptions that are difficult to realize in reality. Hardware-based attestation schemes [14] have stronger safety, yet, security hardware is complex and costly. The third hybrid scheme [15] combines software and hardware. They are designed to reduce the hardware security features that are needed for secure remote attestation. All of these techniques work only in a one-to-one environment. Therefore, these solutions are difficult to scale.

Current one-to-one RA schemes can not attest to a number of devices in a short period. To tackle this issue many researchers have started developing attestation schemes, which we refer to as Collective Remote Attestation (CRA) scheme.

Asokan *et al*. proposed SEDA [16], the first CRA technique, in 2015. The idea is that the verifier performs attestation over an overlay of spanning trees. However, it is assumed that during the attestation phase, the network topology is static. Ambrosin *et al*. presented a new multisignature scheme, called SANA [10]. It is enabled to aggregate attestation results across the network via untrusted aggregators. However, SANA introduces a severe computation overhead for low-end embedded devices in the IoT.

Ibrahim *et al*. proposed DARPA [17] as an attempt to improve SEDA [16] to detect physical attacks. SCAPI [18] is an improved version of DARPA [17]. While DARPA [17] and SCAPI [18] are capable of defending against stronger attackers, they require provers to be available at all times to receive periodically exchanged messages.

To support device mobility during the attestation phase, Kohnhauser *et al*. proposed SALAD [19] for highly dynamic swarm topologies. In addition, SARA [20] enables asynchronous verification of a large number of interconnected IoT devices. However, SARA is restricted to small IoT services.

In recent years, some researchers have also proposed several blockchain-based CRA schemes [21]. These solutions are capable of providing trusted and distributed mechanism for IoT environments [22]. However, blockchain brings in a lot of calculation and storage expenses, which is difficult for low-end devices.

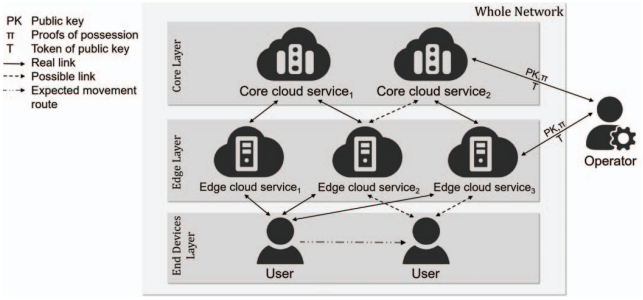


Fig. 1: System model and topology in a cloud-edge-end cooperation environment.

The above-mentioned solutions are for the low-end devices in the IoT. We apply the idea of using multisignatures for batch attestation of devices in SANA [10] to this paper for a cloud-edge-end cooperation environment.

III. System and Adversary Model

A. System Model and Entities

There are four major categories of entities in the SLCSA. The whole network can be divided into three layers: Core Layer, Edge layer, and End Devices Layer. When attesting, a variable topology is generated with the user as the root node, as depicted in Fig. 1.

- *The Network Operator (**O**):* **O** is the operator of the entire network and is responsible for the deployment and maintenance of all devices in the network. **O** is required to initialize all devices before proceeding with the attestation protocol. **O** is also a trusted authority (TA) responsible for distributing public key tokens for each cooperative service to be verified, through a key registration protocol.
- *The end devices, i.e., Users (**U**$_i$):* In a cloud-edge-end cooperation environment, the end device is the user of the cloud platform. These devices may have high mobility. As a result, there are multiple edge and cloud servers to provide them with services. Users act as verifiers and send attestation requests to the cloud platform before uploading their data to ensure the security of their data. After receiving the attestation signatures from cooperative services, users aggregate the signatures to quickly verify that the cooperative service is in the correct state.
- *The edge cloud services, i.e., Provers (**P**$_i$):* As the primary provers in a cloud platform, edge cloud services are typically provided by edge servers deployed near the user, which are powerful devices with greater computation ability and storage capacity than the users. Users usually upload the data to the edge cloud for processing first. They are also responsible for forwarding attestation requests from users and signature results from the upper core cloud services. They generally do not have enough memory and processors to implement complex operations, such as deep learning. At this point, they pre-process the data and send it to the core cloud services.
- *The core cloud services, also, Provers (**P**$_i$):* The core cloud services are also part of what makes up provers, which are generally farther away from the user but have greater computing and storage capacity than edge cloud services. Core cloud services and edge cloud services are complementary. When there is user data that the edge cloud cannot handle, it can be handed over to the core cloud.

In our proposed scheme, **O** assigns the appropriate cooperative services to the users. The services (i.e., provers) assigned to users are variable. However, it doesn't mean that the service assigned during the attestation process is changeable.

B. Adversary Model

An adversary's primary goal is to cause damage to the cooperative service provided by the cloud platform. In this way, it steals the data uploaded by the user or terminates the normal operation of the service. Moreover, the adversary **A** does an effort not to be detected by the attestation mechanism. We, along with many other collective remote attestation schemes, consider only software adversaries ($\mathbf{Adv}_{SW}$) as defined in [23]. This type of adversary has the ability to run malicious code or firmware on a device.

We do not consider Mobile Software Adversary ($\mathbf{Adv}_{MSW}$), i.e., adversaries that can hide the traces of their intrusion. We also disregard physical adversaries, i.e., adversaries are capable of mounting side-channel attacks or capturing the device(s). They can extract the cryptographic material.

In the actual situation, users of the cloud platform may be controlled by adversaries. Denial-of-service (DoS) attacks are achieved by continuously sending attestation requests to the cloud platform. In general, DoS attacks are hard to mitigate. This attack is outside of our current scope.

C. Security Goals

Following the collective attestation literature, we list the following goals that we aim to achieve through the SLCSA protocol. A secure collective remote attestation protocol should possess the following goals:

- *Successful Attestation:* The primary goal of SLCSA is to enable users to successfully verify all cooperative services provided by the cloud platform. The user can verify the operation status of all the cooperative services assigned to them.
- *Unforgeability and Freshness:* Unforgeability ensures that the attestation result is a true reflection of the state of the cooperative service. This attestation result cannot be tampered with by the adversary. Moreover, the freshness of the attestation result can prevent the adversary from replay attacks.
- *Scalablity:* Since the cooperative services assigned to the users are not fixed and may change, the SLCSA solution must support the joining or leaving of cooperative services.
- *Information of the Individual Services:* In SLCSA, users should not only have the ability to verify all cooperative

services but also have the ability to find out the compromised service(s).
- *Parallel Execution:* SLCSA must support multiple parallel or overlapping attestation protocol instances.

D. Security Assumptions

We assume that all $\mathbf{P}_i$ have a Trusted Execution Environment (TEE) to correctly achieve secure remote attestation. A software integrity measurement mechanism is required, while measurement and generated reports cannot be tampered with by **A**. This report can demonstrate reliability from the underlying hardware to the service. TEE can also ensure that the $\mathbf{P}_i$'s private key used for signing is not leaked and that the protocol runs correctly. For our solution, we assume that each $\mathbf{P}_i$ has a secure enclave that can be successfully verified by the network operator **O**.

IV. Preliminaries and Notations

Let $|M|$ denote the number of elements in a finite set *M*. If n is an integer (or a bit-string) $|n|$ indicates the bit-length of n. Let $m \xleftarrow{R} M$ denote the assignment of a uniformly sampled element of *M* to variable m. Furthermore, let $\{0, 1\}^l$ be the set of all bit-strings of length l. If *E* is some event (e.g., the result of a security experiment), then **Pr**[*E*] denotes the probability that *E* occurs. Probability $\epsilon(l)$ is called *negligible* if, for all polynomials f, $\epsilon(l) \leq 1/f(l)$ for all sufficiently large $l \in \mathbb{N}$. Let **F** be a probabilistic algorithm. Then $y \leftarrow \mathbf{F}(x)$ means that on input x, **F** assigns its output to variable y.

V. Scheme Description

Our signature scheme is based on the BLS multisignature. The details of the signature scheme are described in Section V-A. Furthermore, in Section V-B, we introduce the specific process of the proposed protocol.

A. Pairing-Based Signature Schemes

In the following part, we describe the multisignature scheme used in this paper. Since the user's location may change in our scenario, the cooperative services to be used are not fixed. There may also be a loss of response during the signature process. To facilitate the increase and removal of services and the verification of the final aggregate signature, we choose to use a pairing-based aggregated multisignature.

Our signature scheme requires: 1) Let $\mathbb{G}_1$,$\mathbb{G}_2$,$\mathbb{G}_t$ be multiplicative groups of prime order *p* with generators g_1, g_2, g_t, respectively, with an efficiently computable bilinear pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_t$ so that $e(g_1^x, g_2^y) = g_t^{xy}$ for all x, $y \in \mathbb{Z}_p$; 2) a hash function $\mathbf{H}_0 : \{0, 1\}^* \rightarrow \mathbb{G}_1$.

Parameters Initialization. PIn(ω) initializes a multiplicative bilinear group (p, $\mathbb{G}_1$, $\mathbb{G}_2$, $\mathbb{G}_t$, e, g_1, g_2, g_t) $\leftarrow \Gamma(\omega)$ and outputs $par \leftarrow$ (p, $\mathbb{G}_1$, $\mathbb{G}_2$, $\mathbb{G}_t$, e, g_1, g_2, g_t).

Key Generation. KGe(par) generates the public key $pk_i \leftarrow g_2^{sk_i}$ by randomly choosing a secret key $sk_i \xleftarrow{R} \mathbb{Z}_p$ and outputs (sk_i, pk_i).

Key Aggregation. The key aggregation algorithm **KAg**(pk_n, pk_m) for two individual public keys pk_n, pk_m, outputs $apk \leftarrow pk_n \cdot pk_m$.

Key Deletion. The key deletion algorithm **KDe**(apk, pk_i) for the aggregate public key apk and the individual public key pk_i, which is already aggregated in apk, outputs $apk' \leftarrow apk/pk_i$.

Signing. This requires only one round of interactive communication. The signing algorithm **SIG**(par, sk_i, pk_i, m) computes a signature σ_i on the message $m : \sigma_i \leftarrow \mathbf{H}_0(m)^{sk_i}$, and outputs σ_i.

Signature Aggregation. Aggregating several individual signatures σ_1, σ_2, ... , σ_n can be done by computing $\sigma \leftarrow \sigma_1 \cdot \sigma_2 \cdots \sigma_n$. The signature aggregation algorithm **SAg**($\{\sigma_1$, σ_2, ... , $\sigma_n\}$) outputs σ.

Verification. The verification algorithm **VER**(par, apk, m, σ) verifies an aggregate signature σ under message m by computing $e(\sigma, g_2) = e(\mathbf{H}_0(m), apk)$. If so, then outputs 1, otherwise outputs $\perp$.

B. Remote Attestation Protocol Based on Multisignature

Our attestation protocol consists of five main phases: 1) each entity in the whole network is initialized in a trusted environment by the network operator; 2) cooperative services request public key tokens from the network operator; 3) the user of the cloud platform verifies the validation of the token and computes the aggregated public key; 4) After broadcasting a clear request to the edge cloud, the user can send an attestation request to the cooperative services for verifying and the cooperative service computes the signature based on the received attestation request and its own state; 5) user verifies the aggregated signature using the aggregated public key.

Initialization. The network operator (**O**) first determines a security parameter ω of the entire network and the system parameter par is generated by **PIn**(ω). Then **O** performs a traditional remote attestation with the enclave in each $\mathbf{P}_i$. When a new cooperative service is joined, **O** also conducts a remote attestation with its enclave. After successful attestation, a secure channel is established between **O** and the enclave. Afterwards, each service ($\mathbf{P}_i$) provided by the cloud platform is initialized by **O**, and a legal measurement report R_i is added to each $\mathbf{P}_i$. The report stores the correct parameter information for all software configurations for $\mathbf{P}_i$. Then, **O** provides a unique session identifier q_i for each $\mathbf{U}_i$. It is used to prevent duplicate attestation. Finally, **O** assigns the appropriate cooperative service to $\mathbf{U}_i$, adding the identity id_i of the service that $\mathbf{U}_i$ may use to the $\mathbf{U}_i$'s local storage set S_i. Moreover, the legal measurement report R_i for each $\mathbf{P}_i$ is also added to the $\mathbf{U}_i$'s local set RS_i. Finally, each $\mathbf{P}_i$ initializes its set of session identifiers Q_i to an empty set.

Key registration. To enhance the security of the whole system, all cooperative services have to apply for a public key token T_i by executing a protocol **KeyReg** (see Fig. 2) with **O**. $T_i = \{id_i, pk_i, pk_i', t_{exp}, \sigma_T\}$ contains the expiration

period of the key currently used by the service, and only signatures generated by keys within the expiration period can be verified successfully. In detail, when a new cooperative service is available or when the currently used key expires, in order to obtain the legal key pair used in the signing phase, $\mathbf{P}_i$'s secure enclave first executes the algorithm **KGe**(par) to obtain a new secret and public key pair ($sk_i \xleftarrow{R} \mathbb{Z}_p$, $pk_i \leftarrow g_2^{sk_i}$) based on par. If this service is newly offered or to be reassigned, it will set $pk_i^{'} = \perp$. Otherwise, enclave will assign the value of the last expired public key to $pk_i^{'}$. Enclave generates the current measurement report $R_i^{'}$ and signs it with the newly generated key. Then $\mathbf{P}_i$'s enclave gets π_i ($\pi_i \leftarrow \mathbf{H}_0(R_i^{'})^{sk_i}$) and sends id_i, the new public key pk_i, the expired public key $pk_i^{'}$, as well as the π_i to **O** through the secure channel. **O** then verifies the validity of π_i, and if $e(\pi_i, g_2) = e(\mathbf{H}_0(R_i), pk_i)$, **O** will assign an expiry time $t_{exp} \leftarrow t_{now} + \Delta t$ to this public key according to the actual situation. Afterwards, **O** computes a signature σ_T on id_i, pk_i, $pk_i^{'}$, and t_{exp} using **O**'s secret key sk_O — **SIG**(par, sk_O, pk_O, $id_i|pk_i|pk_i^{'}|t_{exp}$). We use the BLS signature for the signature algorithm here. Of course, since no aggregation is involved in the key registration phase, other secure signature algorithms can be used. Then **O** sends T_i to the corresponding cooperative service. If $pk_i^{'} = \perp$, **O** assigns the service to the users who need it before sending the token T_i. It means that during the key registration phase **O** may update S_i and RS_i of some users. Once $\mathbf{P}_i$ obtains the token, it will broadcast its identity id_i and T_i in the end devices layer after validating the token.

Attestation preparation. As shown in Fig. 2, $\mathbf{U}_i$ computes the aggregate public key in advance by the protocol **Prepar** in the attestation preparation phase. In the subsequent attestation process, $\mathbf{U}_i$ can quickly perform batch attestation of multiple $\mathbf{P}_i$. Upon receiving the public key token information (id_i, T_i) broadcast by the cooperative service, $\mathbf{U}_i$ accepts this message if id_i is in its local set S_i. $\mathbf{U}_i$ first verifies if t_{exp} is greater than the current time, and then verifies the signature σ_T in the token by using **O**'s public key pk_O — **VER**(par, pk_O, $id_i|pk_i|pk_i^{'}|t_{exp}$, σ_T). If both are valid, pk_i is added to the locally stored aggregate public key and the set PS_i. The public key information stored in the set PS_i contains the expiry time of these public keys. If $pk_i^{'} = \perp$, $\mathbf{U}_i$ executes **KAg**(apk, pk_i) directly, otherwise, $\mathbf{U}_i$ performs **KDe**(apk, $pk_i^{'}$) first and then **KAg**(apk, pk_i). If **O** removes a cooperative service $\mathbf{P}_i$ from assignment to a user, the user first removes id_i of $\mathbf{P}_i$ from S_i. Then, **KDe**(apk, pk_i) is executed. $\mathbf{U}_i$ also updates the set PS_i synchronously as shown in Fig. 2. At the end of the attestation preparation, $\mathbf{U}_i$ sends a clear request **Clear**(q_i) to the edge cloud. The edge cloud then forwards the request to the core cloud in the upper layer. On receipt of the request, $\mathbf{P}_i$ removes q_i from its set Q_i of session identifiers. $\mathbf{P}_i$ only receive clear requests during the preparation phase.

Attestation signature. The protocol **AttSig** is initiated by $\mathbf{U}_i$. As soon as the user is ready for attestation, software integrity verification can be performed on the edge and core cloud services (see Fig. 3). First of all, $\mathbf{U}_i$ confirms the cooperative services to be verified and their software configuration information based on the set S_i = {id_1, ... , id_n} and RS_i = {R_1, ... , R_n}. The set RS_i is then hashed into one single message $R_d \leftarrow \mathbf{H}_2(R_1|$ ... $|R_n)$ and $\mathbf{H}_2$ is an arbitrary hash function $\mathbf{H}_2 : \{0, 1\}^* \rightarrow \mathbb{Z}_p$. Next, $\mathbf{U}_i$ delivers the attestation request $Req = \{N, q_i, S_i, R_d\}$ to the edge cloud services that may be used. N is a random challenge value chosen by $\mathbf{U}_i$; q_i is the session identifier; S_i is the set of services assigned to $\mathbf{U}_i$ and R_d is used as part of the default message. After sending the attestation request, $\mathbf{U}_i$ chooses a reasonable time $t_{end} \leftarrow t_{now} + \delta t$ to stop receiving responses based on network latency, bandwidth, number of services, and other factors. When $\mathbf{P}_i$ receives the attestation request, it first checks whether its id_i is in S_i. Discard the request if $id_i \notin S_i$, otherwise ($id_i \in S_i$) check if the session identifier q_i of the current attestation request exists in the set Q_i. If $q_i \in Q_i$, the request is discarded, if not ($q_i \notin Q_i$), q_i can be appended to Q_i. The session identifier is stored in the set Q_i until the next attestation preparation and q_i is necessary to protect the network against replay attacks. If the service is an edge cloud service, the request will be broadcast to the core cloud service in the upper layer. After that, it proceeds to the next phase.

As a next step in the protocol, each $\mathbf{P}_i$'s enclave generates its own measurement report $R_i^{'}$. If $R_i^{'}$ is a benign configuration report (i.e., $R_i^{'} = R_i$), enclave creates a BLS signature σ_i over the default message $M = R_d|N|q_i$ — **SIG**(par, sk_i, pk_i, $R_d|N|q_i$). Otherwise ($R_i^{'} \neq R_i$), instead of signing, enclave adds its own public key pk_i to the set B_i. Then the signature σ_i and the set B_i of the edge cloud service are sent directly to $\mathbf{U}_i$ along with id_i. Meanwhile, the response from the core cloud service is forwarded to $\mathbf{U}_i$ through the edge cloud.

Signature verification. As shown in Fig. 3, when $\mathbf{U}_i$ receives an attestation response, it first checks the validity of the public key pk_i corresponding to the signature σ_i in the response. $\mathbf{U}_i$ checks whether the public key pk_i has expired. If the verification is not successful then the response is discarded. After t_{end}, $\mathbf{U}_i$ no longer receives responses and aggregates all responses received. For all the signatures σ_1, ... , σ_n, which are signed over the same default message $M = R_d|N|q_i$, they can be aggregated into a single BLS signature σ_M using the algorithm **SAg**({$\sigma_1, \sigma_2, ..., \sigma_n$}). For all the set $B_1, ..., B_n$, $\mathbf{U}_i$ simply computes $B_M \leftarrow B_1 \cup ... \cup B_n$. For $\mathbf{P}_i$ whose id_i is in $\mathbf{U}_i$'s locally stored set S_i, but from which $\mathbf{U}_i$ has not received a response, $\mathbf{U}_i$ stores its public key pk_i in the set K. After that, $\mathbf{U}_i$ computes the temporary aggregate public key apk_M for this attestation depending on B_M and K — **KDe**(apk, pk_i) for all $pk_i \in B_M \cup K$. Finally, $\mathbf{U}_i$ uses apk_M to verify σ_M — **VER**(par, apk_M, M, σ_M). If the verification succeeds and $B_M \cup K = \varnothing$, $\mathbf{U}_i$ concludes that all the cooperative services are trustworthy. If $B_M \cup K \neq \varnothing$, $\mathbf{U}_i$ also learns the identity of all bad services according to B_M and K.

VI. Performance Evaluation

In this section, we evaluate the computation, communication, and storage overhead of SLCSA. We consider cryp-

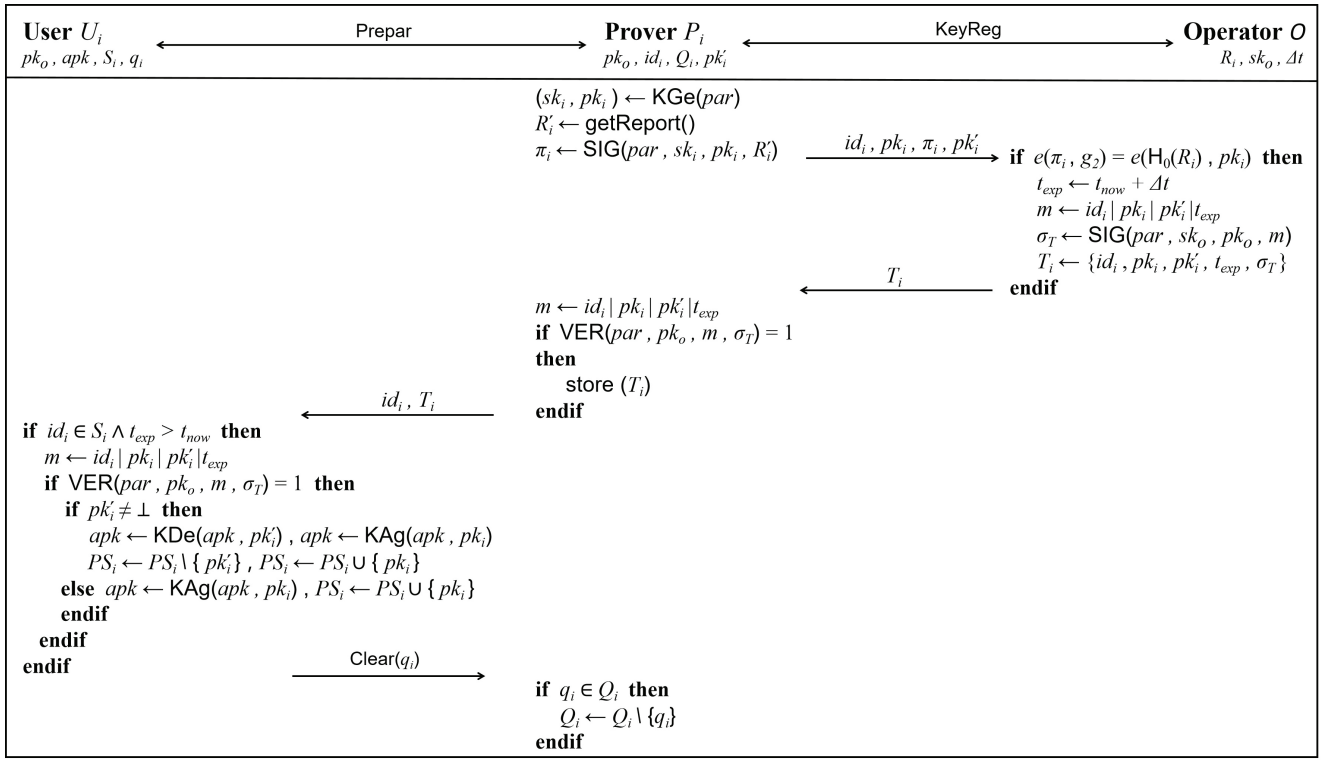


Fig. 2: **Protocol KeyReg** and **Prepar**.

tographic operations, including exponentiations in $\mathbb{G}_1$, $\mathbb{G}_2$, $\mathbb{G}_t$, and pairing operations, that are denoted as EX_1, EX_2, EX_t and P, respectively. In addition, we denote HA_1 as the operation, hashing to $\mathbb{G}_1$. Multiplications in $\mathbb{G}_1$ and $\mathbb{G}_2$ are denoted as MU_1 and MU_2, respectively. $|\mathbb{G}_1|$, $|\mathbb{G}_2|$, and $|\mathbb{G}_t|$ are denoted as the size of corresponding group elements. The signature scheme used by $\mathbf{P}_i$ to participate in remote attestation is BLS multisignature as adopted in [11]. In addition, the signature scheme used by **O** to sign the public token T_i is BLS as adopted in [24]. We denote n as the number of total cooperative services to be verified by a user $\mathbf{U}_i$.

We simulate SLCSA on an Intel NUC10i7FNH with an Intel Core i7-10710U 1.10 GHz processor and 16 GB memory. We use the Intel SGX technology [12] to implement a Trusted Execution Environment on the cloud side, which provides hardware-assisted isolation of system components with real-time execution. Isolation is the basis for protecting crucial components from accidental access by other possibly rogue components. We put part of the code and data of the attestation protocol into the SGX Enclave so that only trusted components can access the protocol's secret data such as the code of **KeyReg** protocol, **AttSig** protocol, and the BLS secret key sk_i.

Our SLCSA scheme implementation is simulated over the BN254 pairing-friendly elliptic curve [25] in Miracl Core library [26], which provides a 128-bit security level. The size of $\mathbb{G}_1$, $\mathbb{G}_2$, and $\mathbb{Z}_p$ in the BN254 curve is 512 bit, 1024 bit, and 256 bit, respectively.

A. Key Registration

The key registration mainly involves interactions between $\mathbf{P}_i$ and **O**. For $\mathbf{P}_i$, it generates its own BLS key pair and computes the signature π_i over the current measurement report. Then, it sends the key registration request to **O** and verifies the response. $\mathbf{P}_i$ needs to conduct one EX_1, one EX_2, two HA_1 and two P. For **O**, it verifies the correctness of π_i and generates a BLS signature on the public token T_i. As a result, **O** needs to conduct two P, two HA_1, and one EX_1 for each $\mathbf{P}_i$. According to Table I, the computation overhead is 3.79 ms for $\mathbf{P}_i$ and 3.28 ms for **O** to negotiate a token, which is feasible in the real world.

TABLE I: Performance of SLCSA algorithms.

Function	Run-time (ms)
KeyGen	0.72
SigGen	0.47
SigVerify	2.77
TokenGen	0.51
TokenVerify	2.60

For the communication overhead, **O** receives a request $Req = \{id_i, pk_i, \pi_i, pk_i^{'}\}$ sent by $\mathbf{P}_i$ and returns a token T_i = $\{id_i, pk_i, pk_i^{'}, t_{exp}, \sigma_T\}$. Our SLCSA implementation has

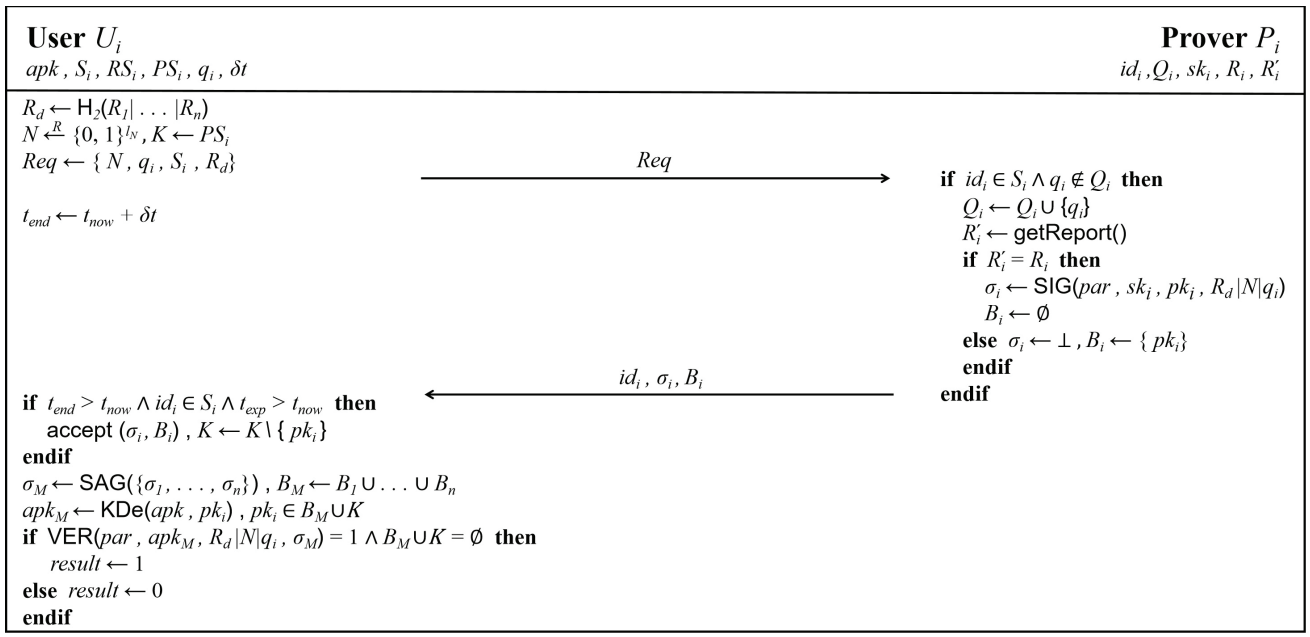


Fig. 3: **Protocol AttSig** and **SigVer**.

a signature size of 64 bytes. In addition, the identity id_i of $\mathbf{P}_i$ is 2 bytes, the public key is 128 bytes, and the time value is 8 bytes. Therefore, a request Req consists of 322 bytes and a response consists of 330 bytes.

For the storage overhead at $\mathbf{P}_i$, it stores the new key pair (sk_i, pk_i), the expired public key pk_i', the public key pk_O, and the token T_i as shown in Table II. Since **O** is the operator of the network, we do not consider its storage overhead.

TABLE II: Complexity analysis of key registration.

	Computation	Storage
$\mathbf{P}_i$	$EX_1 + EX_2 + 2HA_1 + 2P$	$\|\mathbb{Z}_p\|+\|\mathbb{G}_1\|+3\|\mathbb{G}_2\|$ $+\|id_i\|+\|t_{exp}\|$
O	$EX_1 + 2HA_1 + 2P$	$*$

B. Attestation Preparation

$\mathbf{U}_i$ obtains the aggregate public key used to verify the aggregate signature through **Prepar** protocol. Once $\mathbf{U}_i$ receives the public key token T_i for the cooperative service, it will first verify the validity of the token, which results in a linearly increasing computational overhead with the number of $\mathbf{P}_i$ as shown in Fig. 4. This overhead is a maximum of $n \cdot (HA_1 + 2P)$. We denote n_T as the number of cooperative services sending tokens in the preparation phase. So, $\mathbf{U}_i$ needs to conduct $n_T \cdot (HA_1 + 2P)$. Afterwards, $\mathbf{U}_i$ computes an aggregate public key and the run-time for aggregating keys is shown in Fig. 5. During this phase, $\mathbf{P}_i$ has no computational overhead.

For the communication overhead, the edge cloud service broadcasts its token T_i at the end device layer while the core cloud service sends the token to the edge cloud at the lower layer. Edge cloud forwards tokens from the upper layer to broadcast in the lower layer. At the end of **Prepar** protocol, $\mathbf{U}_i$ sends a **Clear**(q_i) request to $\mathbf{P}_i$ and we set q_i to 2 bytes. $\mathbf{P}_i$ makes changes to the set Q_i that stores q_i based on this request.

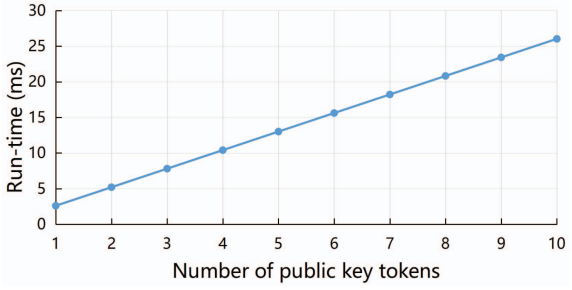


Fig. 4: Run-time of verifying public key tokens.

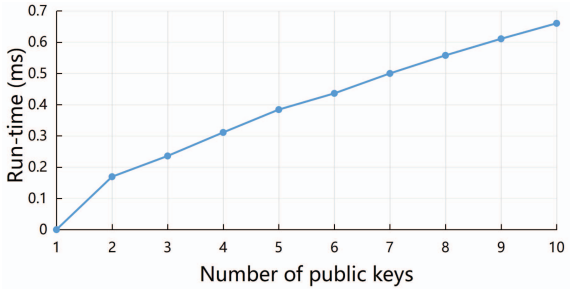


Fig. 5: Run-time of aggregating public keys.

As shown in Table III, for the storage overhead at $\mathbf{P}_i$, it stores a set Q_i of unique global session identifiers. This set can help $\mathbf{P}_i$ resist replay attacks and also prevent $\mathbf{P}_i$ from repeatedly computing signatures. Since $\mathbf{U}_i$ needs to store the set S_i of $\mathbf{P}_i$'s identity, the set PS_i of $\mathbf{P}_i$'s public key, and pk_i's expiry time, the storage overhead for $\mathbf{U}_i$ is mainly affected by n. Besides, it stores the session identifier q_i, the public key of $\mathbf{O}$, and the aggregate public key apk, which has a constant size of 128 bytes.

TABLE III: Complexity analysis of attestation preparation.

	Computation	Storage
$\mathbf{P}_i$	$*$	$\lvert id_i\rvert+\lvert Q_i\rvert$
$\mathbf{U}_i$	$n_T \cdot (HA_1 + 2P + MU_2)$	$(n+2)\lvert\mathbb{G}_2\rvert+\lvert q_i\rvert$ $+n(\lvert id_i\rvert+\lvert t_{exp}\rvert)$

C. Attestation Signature and Signature Verification

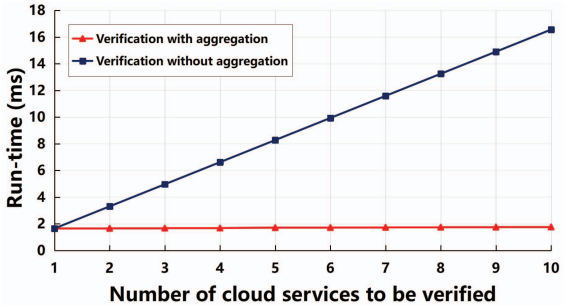
The two phases mainly involve communication between $\mathbf{U}_i$ and $\mathbf{P}_i$. For $\mathbf{U}_i$, it generates R_d based on the set RS_i using a hash function $\mathbf{H}_2$ and chooses a random nonce N. We consider these computational overheads are negligible compared to the cryptographic operations on the curve. When an attestation request is received, $\mathbf{P}_i$ generates a signature σ_i and a set B_i based on the software integrity measurement report. We put this part of the attestation protocol code into enclave so that $\mathbf{P}_i$ can generate σ_i and B_i correctly. Therefore, $\mathbf{P}_i$ only needs to conduct one EX_1 and one HA_1 for a request and the computational overhead for generating a response is just 0.40 ms. For $\mathbf{U}_i$, it aggregates the received signatures and generates a temporary aggregate public key apk_M. $\mathbf{U}_i$ uses apk_M to verify the aggregate signature. We denote n_M as the number of cooperative services participating in generating the BLS multisignature. Therefore, $\mathbf{U}_i$ still needs to conduct one HA_1 and two P.

As shown in Fig. 6(a), we compare the overhead of signature verification with and without aggregation. $\mathbf{U}_i$ can verify 10 cooperative services in just 1.77 ms with aggregation and this advantage expands as the number of cooperative services to be verified increases. In Table IV, we compare the computation overhead of SLCSA and SANA [10] during the attestation phase. The computation overhead in this phase mainly consists of signature generation and signature verification. For $\mathbf{U}_i$, our solution has the additional computation overhead of aggregating signatures. However, when the number of signatures is relatively small, this overhead is almost negligible compared to the overhead of verifying the signature. In Fig. 6(b), we assume that all services provided by the cloud platform are working properly (i.e., no cooperative services are compromised by adversaries) and show a comparison of the runtime in the attestation phase using the SLCSA and SANA schemes. Disregarding the impact of network latency, it is seen that our solution is able to verify multiple edge and core cloud services more quickly in the cloud-edge-end cooperation environment.

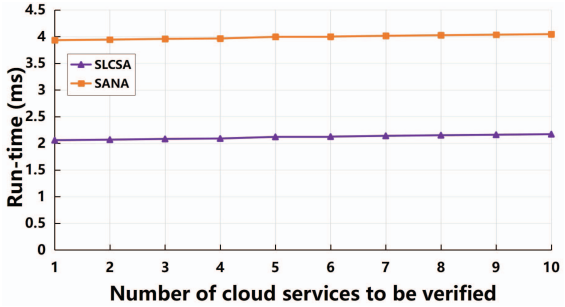
For the communication overhead, $\mathbf{U}_i$ starts the attestation protocol by sending a request $Req = \{N, q_i, S_i, R_d\}$ to the edge cloud. This request is then forwarded by the edge cloud to the core cloud. In our SLCSA implementation, we set N to $l_N = 32$ bytes and R_d is a 32 bytes hash value. So, the request is $66 + 2n$ bytes. After generating a signature, $\mathbf{P}_i$ returns a response including σ_i, B_i, and id_i. As a result, the response is a maximum of 68 bytes.

For the storage overhead at $\mathbf{P}_i$, it stores id_i, sk_i, and Q_i. Moreover, it also stores the correct measurement report R_i, which can be a 32 bytes hash value. In Table V, the storage size of $\mathbf{U}_i$ grows with the number of $\mathbf{P}_i$ to be verified.

In summary, since the BLS multisignature requires only one communication round, this greatly reduces the communication



(a) Run-time of verification with/without aggregation.



(b) Comparison of the runtime of the attestation phase between SLCSA and SANA [10].

Fig. 6: Performance evaluation of the attestation phase.

TABLE IV: Comparison of the computation overhead of the attestation phase between SLCSA and SANA [10].

	$\mathbf{P}_i$	$\mathbf{U}_i$
SLCSA	$EX_1 + HA_1$	$(n_M-1)MU_1 + HA_1 + 2P$
SANA [10]	$EX_1 + 2HA_1 + 2P$ $+mMU_1$[a]	$HA_1 + 2P$

[a] m denotes the number of neighbor nodes of $\mathbf{P}_i$.

TABLE V: Complexity analysis of attestation signature and signature verification.

	Computation	Storage
$\mathbf{P}_i$	$EX_1 + HA_1$	$\lvert id_i\rvert+\lvert Q_i\rvert+\lvert\mathbb{G}_1\rvert$ $+\lvert\mathbb{Z}_p\rvert+\lvert R_i\rvert$
$\mathbf{U}_i$	$(n_M-1)MU_1 + HA_1 + 2P$	$(n+1)\lvert\mathbb{G}_2\rvert+\lvert B_M\rvert+\lvert q_i\rvert$ $+n(\lvert\mathbb{G}_1\rvert+\lvert id_i\rvert+\lvert t_{exp}\rvert)$

overhead of $\mathbf{U}_i$ and $\mathbf{P}_i$. More importantly, the use of multisignature can greatly improve the efficiency of remote attestation. Furthermore, we use the Intel SGX technology to implement a trusted execution environment, which is supported by most Intel CPUs. It demonstrates the feasibility of our scheme in real-world applications.

VII. Security of SLCSA

The security goal of SLCSA is for a user to declare the cloud platform to be trustworthy, if and only if all the cooperative services provided by the cloud platform have the correct software configuration. We formalize this goal as a security experiment $\mathbf{Exp_A}$ among an adversary $\mathbf{A}$, $\mathbf{U}_i$, $\mathbf{P}_i$, and $\mathbf{O}$. We present five possible attack strategies that could be implemented by $\mathbf{A}$ and analyze the possibility of success for each of these strategies. In the security experiment $\mathbf{Exp_A}$, $\mathbf{A}$ attacks cooperative services in the cloud platform and modifies the program code of at least one service $\mathbf{P}_A$. In addition, $\mathbf{A}$ can eavesdrop on, delete, and modify any message sent from $\mathbf{P}_A$. After a polynomial number (in terms of the security parameters l_N, l_s, and l_c) of steps by $\mathbf{P}_A$, $\mathbf{U}_i$ outputs its decision $result = 1$ indicating that attestation of the services to be used is successful, or $result = 0$ otherwise. The result of the experiment is defined as the output of $\mathbf{U}_i$, i.e., $\mathbf{Exp_A} = result$. A secure collective remote attestation scheme is defined as follows:

Definition 1 (Computational co-Diffie-Hellman Problem)**.** *For a groups $\mathbb{G}_1 = \langle g_1 \rangle$, $\mathbb{G}_2 = \langle g_2 \rangle$, of prime order p, define $\mathbf{Adv}^{\mathrm{co-CDH}}_{\mathbb{G}_1,\mathbb{G}_2}$ of an adversary $\boldsymbol{A}$ as*

$$\boldsymbol{Pr}\ [y = g_1^{\alpha\beta} : (\alpha, \beta) \xleftarrow{\mathrm{R}} \mathbb{Z}_p^2, y \leftarrow \boldsymbol{A}(g_1^{\alpha}, g_1^{\beta}, g_2^{\beta})],$$

where the probability is taken over the random choices of $\boldsymbol{A}$ and the random selection of (α, β). $\boldsymbol{A}(\tau, \epsilon)$-breaks the **co-CDH** *problem if it runs in time at most τ and has $\mathbf{Adv}^{\mathrm{co-CDH}}_{\mathbb{G}_1,\mathbb{G}_2} \geqslant \epsilon$.* **co-CDH** *is (τ, ϵ)-hard if no such adversary exists.*

Theorem 1 (Security of BLS Multisignature Scheme)**.** *BLS multisignature scheme is an unforgeable multisignature scheme under the hardness of the* **co-CDH** problem *(Definition 1) in the random-oracle model.*

Proof of Theorem 1. Owing to the length of the proof of Theorem 1, we do not describe it in detail in this paper. The reader may refer to [11] for the security proof of BLS multisignature scheme. □

Definition 2 (Secure Collective Remote Attestation)**.** *A collective remote attestation scheme is if* $\mathbf{Pr}\ [result = 1 | \boldsymbol{Exp_A}(1^l) = result]$ *is negligible in $l = f(l_N, l_{bls}, l_{sign})$, where the function f is polynomial in l_N, l_{bls}, and l_{sign}.*

Theorem 2 (Security of SLCSA)**.** *SLCSA is a secure collective remote attestation scheme (Definition 2) if the underlying BLS multisignature scheme is not forgeable (Theorem 1).*

Proof (sketch) of Theorem 2. $\mathbf{U}_i$ returns $result = 1$, i.e., all the edge and core cloud services to be used are running a benign software configuration initialized by the operator $\mathbf{O}$ of the network only if $\mathbf{VER}(par, apk_M, M, \sigma_M) \wedge B_M \cup K = \varnothing$, where par is the system signature parameter and apk_M is the BLS aggregate public key of all cooperative services that participate in generating the BLS multisignature. σ_M is the aggregate signature over message M and $M = R_d|N|q_i$ is the default signature message. R_d is the hash digest of the set S_i, N is a random nonce, and q_i is the unique global session identifier of $\mathbf{U}_i$. B_M and K are the sets of public keys that did not participate in generating the BLS multisignature. To avoid detection of a compromised service $\mathbf{P}_A$, $\mathbf{A}$ can use one of the following strategies: 1) $\mathbf{A}$ modifies the software configuration of $\mathbf{P}_A$ but does not tamper with the response from $\mathbf{P}_A$; 2) $\mathbf{A}$ uses an old response $(id_i, \sigma_{old}, B_{old}, T_i)$ previously generated by $\mathbf{P}_A$ on the benign measurement report; 3) $\mathbf{P}_A$ discards the correct response and generates a BLS signature over the good measurement report as the response; 4) $\mathbf{P}_A$ forges a public key token T_A of pk_A to send to $\mathbf{U}_i$ during **Prepar** protocol and signs the default signature message M using $\mathbf{A}$'s secret key sk_A; 5) $\mathbf{P}_A$ forges a pair of keys and sends the public key to $\mathbf{O}$ to request a valid public key token.

We start with strategy 1). According to the assumption made in Section III-D, $\mathbf{A}$ cannot tamper with the software integrity measurement mechanism and the measurement report on $\mathbf{P}_A$. In addition, $\mathbf{P}_A$ has the security hardware that provides a TEE to ensure that the $\mathbf{P}_A$'s secret key sk_i is not leaked and that the protocol code runs correctly. Consequently, $\mathbf{P}_A$ does not sign the default message M, but adds $\mathbf{P}_A$'s pk_i to the set B_M. If the underlying integrity measurement mechanism that generates the measurement report can detect this modification (i.e., $R_i^{'} \neq R_i$), $\mathbf{U}_i$ will always return $result = 0$.

Next, we consider strategy 2), where $\mathbf{A}$ uses an old response of $\mathbf{P}_A$. The old signature σ_{old} over the old default signature message $M_{old} = R_d|N_{old}|q_i$ as well as the old set B_{old} are sent to $\mathbf{U}_i$. B_{old} may be the empty set and T_i may still be valid. However, $\mathbf{VER}(par, apk_M, M, \sigma_M)$ will be equal to 1 only if $N = N_{old}$, which is negligible in l_N, where l_N represents the size of the nonce. Consequently, $\mathbf{U}_i$ will always return $result = 0$.

According to strategy 3), $\mathbf{A}$ may generate a BLS signature over the default signature message $M = R_d|N|q_i$. Since N is a fresh nonce sent by $\mathbf{U}_i$, the probability of $\mathbf{A}$ signing M is negligible in l_N. Consequently, as BLS multisignature scheme is unforgeable according to Theorem 1, the probability of forging such a signature is negligible in l_{bls}, where l_{bls} represents the security parameter of multisignature scheme.

Then, we consider strategy 4), where $\mathbf{A}$ forges a public key token T_A of (sk_A, pk_A). After receiving the token T_i sent from $\mathbf{P}_i$, $\mathbf{U}_i$ verifies whether the signature σ_T signed by $\mathbf{O}$ in the token is valid. This means that $\mathbf{A}$ must forge σ_T to make $\mathbf{U}_i$ accept. Owing to the selective forgery resistance of the signature scheme, the probability of forging such a signature is negligible in l_{sign}, where l_{sign} represents the security parameter of the signature scheme selected by $\mathbf{O}$.

Finally, we consider strategy 5), according to the assumption

made in Section III-D, each $\mathbf{P}_i$ has a secure enclave. During the key registration phase, $\mathbf{O}$ communicates with the enclave over a secure channel and checks the measurement report before issuing the token. Because of the isolation of the enclave, $\mathbf{A}$ cannot tamper with enclave's internal protocol code and is unable to send his forged key to $\mathbf{O}$ through the enclave. Therefore, it is impossible for $\mathbf{A}$ to obtain a valid token.

Therefore, the probability of $\mathbf{A}$ making $\mathbf{U}_i$ return $result = 1$, when it maliciously modifies the software configuration of at least one service $\mathbf{P}_i$ provided by the cloud platform, is negligible in l_N, l_{bls}, and l_{sign}. □

VIII. Conclusion

In this paper, we propose SLCSA, the first practical and secure attestation protocol for verifying multiple cooperative services in cloud-edge-end cooperation environments. The SLCSA scheme improves the efficiency of attestation by using a BLS multisignature to accelerate signature verification. We demonstrate the efficiency of the SLCSA scheme through a comprehensive performance evaluation. We perform security analysis to demonstrate the security of the SLCSA against software-only attacks. We demonstrate the feasibility of our scheme by implementing it using the SGX technology, which is currently supported by the majority of Intel CPUs. In future, we will explore collective remote attestation in cloud-edge-end cooperative environments. We can incorporate the "heartbeat" protocol [17] into our scheme to optimize the SLCSA for defending against more powerful physical adversaries.

Acknowledgment

The work was supported in part by the National Natural Science Foundation of China (62272002, 62202005), in part by the Excellent Youth Foundation of Anhui Scientific Committee (2108085J31), in part by the University Synergy Innovation Program of Anhui Province (GXXT-2022-049).

References

[1] Radouan Ait Mouha. Internet of things (iot). *Journal of Data Analysis and Information Processing*, 9(2):77–101, 2021.

[2] Rajkumar Buyya, Satish Narayana Srirama, Giuliano Casale, Rodrigo Calheiros, Yogesh Simmhan, Blesson Varghese, Erol Gelenbe, Bahman Javadi, Luis Miguel Vaquero, Marco AS Netto, et al. A manifesto for future generation cloud computing: Research directions for the next decade. *ACM computing surveys (CSUR)*, 51(5):1–38, 2018.

[3] Fengqun Wang, Jie Cui, Qingyang Zhang, Debiao He, Chengjie Gu, and Hong Zhong. Blockchain-based lightweight message authentication for edge-assisted cross-domain industrial internet of things. *IEEE Transactions on Dependable and Secure Computing*, 2023.

[4] Xiaokang Wang, Laurence T Yang, Xia Xie, Jirong Jin, and M Jamal Deen. A cloud-edge computing framework for cyber-physical-social services. *IEEE Communications Magazine*, 55(11):80–85, 2017.

[5] Yinxue Yi, Zufan Zhang, Laurence T Yang, Xiaokang Wang, and Chenquan Gan. Edge-aided control dynamics for information diffusion in social internet of things. *Neurocomputing*, 485:274–284, 2022.

[6] Lu Wei, Jie Cui, Hong Zhong, Irina Bolodurina, and Lu Liu. A lightweight and conditional privacy-preserving authenticated key agreement scheme with multi-ta model for fog-based vanets. *IEEE Transactions on Dependable and Secure Computing*, 2021.

[7] Lei Ren, Yuanjun Laili, Xiang Li, and Xiaokang Wang. Coding-based large-scale task assignment for industrial edge intelligence. *IEEE Transactions on Network Science and Engineering*, 7(4):2286–2297, 2019.

[8] Trusted Computing Group. Trusted computing platform alliance (tcpa) main specification version 1. 1b [r]. 2001.

[9] Ernie Brickell, Jan Camenisch, and Liqun Chen. Direct anonymous attestation. In *Proceedings of the 11th ACM conference on Computer and communications security*, pages 132–145, 2004.

[10] Moreno Ambrosin, Mauro Conti, Ahmad Ibrahim, Gregory Neven, Ahmad-Reza Sadeghi, and Matthias Schunter. Sana: Secure and scalable aggregate network attestation. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 731–742, 2016.

[11] Mihir Bellare and Gregory Neven. Multi-signatures in the plain public-key model and a general forking lemma. In *Proceedings of the 13th ACM conference on Computer and communications security*, pages 390–399, 2006.

[12] Victor Costan and Srinivas Devadas. Intel sgx explained. *Cryptology ePrint Archive*, 2016.

[13] Jin Cao, Tong Zhu, Ruhui Ma, Zhenyang Guo, Yinghui Zhang, and Hui Li. A software-based remote attestation scheme for internet of things devices. *IEEE Transactions on Dependable and Secure Computing*, 2022.

[14] Michał Kucab, Piotr Boryło, and Piotr Chołda. Remote attestation and integrity measurements with intel sgx for virtual machines. *Computers & Security*, 106:102300, 2021.

[15] Ivan De Oliveira Nunes, Sashidhar Jakkamsetti, Norrathep Rattanavipanon, and Gene Tsudik. On the toctou problem in remote attestation. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, pages 2921–2936, 2021.

[16] Nadarajah Asokan, Ferdinand Brasser, Ahmad Ibrahim, Ahmad-Reza Sadeghi, Matthias Schunter, Gene Tsudik, and Christian Wachsmann. Seda: Scalable embedded device attestation. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 964–975, 2015.

[17] Ahmad Ibrahim, Ahmad-Reza Sadeghi, Gene Tsudik, and Shaza Zeitouni. Darpa: Device attestation resilient to physical attacks. In *Proceedings of the 9th ACM Conference on Security & Privacy in Wireless and Mobile Networks*, pages 171–182, 2016.

[18] Florian Kohnhäuser, Niklas Büscher, Sebastian Gabmeyer, and Stefan Katzenbeisser. Scapi: a scalable attestation protocol to detect software and physical attacks. In *Proceedings of the 10th ACM Conference on Security and Privacy in Wireless and Mobile Networks*, pages 75–86, 2017.

[19] Florian Kohnhäuser, Niklas Büscher, and Stefan Katzenbeisser. Salad: Secure and lightweight attestation of highly dynamic and disruptive networks. In *Proceedings of the 2018 on Asia Conference on Computer and Communications Security*, pages 329–342, 2018.

[20] Edlira Dushku, Md Masoom Rabbani, Mauro Conti, Luigi V Mancini, and Silvio Ranise. Sara: Secure asynchronous remote attestation for iot systems. *IEEE Transactions on Information Forensics and Security*, 15:3123–3136, 2020.

[21] Lukas Petzi, Ala Eddine Ben Yahya, Alexandra Dmitrienko, Gene Tsudik, Thomas Prantl, and Samuel Kounev. {SCRAPS}: Scalable collective remote attestation for {Pub-Sub}{IoT} networks with untrusted proxy verifier. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 3485–3501, 2022.

[22] Muhammad Salek Ali, Massimo Vecchio, Miguel Pincheira, Koustabh Dolui, Fabio Antonelli, and Mubashir Husain Rehmani. Applications of blockchains in the internet of things: A comprehensive survey. *IEEE Communications Surveys & Tutorials*, 21(2):1676–1717, 2018.

[23] Moreno Ambrosin, Mauro Conti, Riccardo Lazzeretti, Md Masoom Rabbani, and Silvio Ranise. Collective remote attestation at the internet of things scale: State-of-the-art and future challenges. *IEEE Communications Surveys & Tutorials*, 22(4):2447–2461, 2020.

[24] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the weil pairing. In *International conference on the theory and application of cryptology and information security*, pages 514–532. Springer, 2001.

[25] Thomas Unterluggauer and Erich Wenger. Efficient pairings and ecc for embedded systems. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 298–315. Springer, 2014.

[26] Miracl core. https://github.com/miracl/core, 2022.