20

The Accountability Strikes Back: Decentralizing the Key Generation in CL-PKC with Traceable Ring Signatures

Varesh Mishra, Aysajan Abidin, and Bart Preneel

COSIC, KU Leuven, Belgium

Abstract

In the context of Federated Learning, it is essential to authenticate the incoming messages from various entities. Many works have utilized Certificateless Public Key Cryptography (CL-PKC) and Signature Schemes. These recent developments focus on the existence of a single Trusted Authority (TA) that can issue partial private signatures truthfully without any malicious intent. This assumption is not pragmatic when multiple competing entities with some assets are involved. Additionally, dependence on a single TA introduces a single point of failure and a center of malice. In this work, we first propose a mechanism for partial key exchange for CL-PKC employing Traceable Ring Signatures. Furthermore, we utilize the existing blockchain or logging infrastructure to extend our model to provide accountability and disincentivization for malicious TAs. The cryptographic tools used in this work can be parallelized to get more efficiency out of these trusted nodes. To evaluate our protocol, we also simulate it to argue for the communication and computation costs.

Keywords: certificateless public key cryptography, distributed learning, accountability.

20.1 Introduction

Federated Learning (FL) is a distinct approach to machine learning in which distributed and decentralized entities train a local model using the localized data and provide updates to the global server to improve the global model. There are many inherent benefits to such an approach as it avoids the unnecessary sharing of raw data from one point to another. It also has numerous advantages in terms of scalability and efficiency. However, this approach generates a complex network structure wherein there is a huge amount of data aggregation to a local node, usually called the aggregator, where local data processing occurs. Due to the complexity of the underlying network structure, enabling key distribution and management is a challenging task. The key management and distribution are essential to implement the necessary data security guarantees. Moreover, it is equally important to handle access control and data authentication from various entities in the network. Without the necessary steps, malicious entities can exploit the vulnerabilities to inflict loss of data and assets. Thus, distributing keys and authenticate various messages from the network entities is of utmost importance. A critical aspect of such networks is that the end devices that collect the data are usually resource-constrained with limited computation power. Therefore, we cannot deploy expensive cryptographic tools for data security.

20.2 Related Work and Contributions

Identity-based Encryption (IBE) [1] [2] allows users to communicate and share secrets and signatures in which the user's public key is derived from a known string associated with the user. However, this requires complete trust in a Trusted Authority (TA), also known as Key Generation Centre (KGC). This introduces a single point of failure that can reveal all the secret keys associated with the system users when compromised. Certificateless Public Key Cryptography (CL-PKC) [3] allows the system users to derive keys using the trusted KGC without the key-escrow problem in IBE. In CL-PKC, the users use KGC to get partial private keys, which they can use to generate their full private keys, reducing the role of KGC. This scheme does not require certificates, but a trust in the KGC is still required. Other issues include securely transmitting the partial private key to the user who requests it. Hierarchical Certificateless Cryptography (HCLC) [4] [5] aims to solve the concerns regarding trust in KGC by introducing a tree-like structure involving a root KGC, lower-level KGCs, and the users. The structure dictated by

HCLC has root KGC at the top, the users as the leaf nodes, and the lower-level KGCs as the intermediate nodes. This model reduces trust in the root KGC but requires a complex setup procedure and has scalability issues when the hierarchy grows. There are concerns regarding communication costs and the security of higher-level KGCs. However, CL-PKC is very efficient and computationally inexpensive for resource-constrained devices. Many recent works propose using CL-PKC in multiple domains, such as Wireless Body Area Networks, VANET authentication, and Federated Learning with privacy and anonymity as a feature [6], [7], [8], [9]. However, all these works rely on a single KGC or TA. In this work, we introduce the usage of multiple and distributed TAs for the full private key generation. The proposed protocol also provides accountability for the key generation process using Traceable Ring Signatures. The TA trying to generate an inconsistent or another key using the partial private key (key replacement) can be disincentivized. Another benefit of using a Traceable Ring Signature is that when the key is successfully generated, the user requesting the key and the TA generating the key only know about the key generation process, introducing uncertainty regarding where it was generated, provided appropriate measures are taken to hide the network traffic. The other participants may become aware that the key has been generated for a user who desires it in specific applications. The user must, however, provide certain publicly known parameters related to TA for consistent encryption, which can be predetermined between the parties if required.

20.3 Preliminaries

20.3.1 Pairing Based Cryptography

Consider an additive group $(G_{\mathbb{1}}, +)$ and a multiplicative group $(G_2, .)$, both finite groups of prime order q. Let us denote P as a generator of $G_{\mathbb{1}}$. A pairing e is a map $e : G_{\mathbb{1}} \times G_{\mathbb{1}} \rightarrow G_2$ such that the following properties hold true:

- The map e is bilinear: given $A, B, C \in G_1$, we have

$$e(A, B + C) = e(A, B).e(A, C),$$

$$e(A + B, C) = e(A, C).e(B, C).$$

  Additionally, for any $x, y \in Z_q^*$, we have $e(xA, yB) = e(A, B)xy = e(xyA, B)$ etc.
- The map e is non-degenerate: $e(P, P) \neq 1$.

- There exists an efficient algorithm such that computation of map e is efficient.

20.3.2 Certificateless Public Key Cryptography (CL-PKC)

CL-PKC is a public key cryptosystem that overcomes the limitations of traditional Public Key Infrastructure and Identity-Based Cryptography (IBC). The main objective of CL-PKC is the elimination of digital certificates (as required in traditional PKI) and the inherent key-escrow problem where the TA possesses the knowledge of the private keys of all the entities. In CL-PKC, a TA or KGC facilitates an entity in the private key generation by generating a partial private key with its secret value. The entity computes its public key using its secret value. It is interesting to observe that an entity can generate its public key before generating the private key.

20.3.3 Traceable Ring Signatures

A ring signature allows the signer to sign a message on behalf of a group of signers and the ability to remain anonymous [10]. Thus, with a ring signature, an entity can ensure that the message was signed by one of the group members but cannot identify the signer amongst the group. Blockchains such as Monero use ring signatures for anonymous transactions [11]. However, in specific applications such as e-voting, total anonymity can help certain entities cast double votes without repercussion. Traceable Ring Signature (TRS) [12] provides functionality to trace a signer's public key in case the signer issues a signature for two messages using the same tag. The tag usually consists of an $issue$ and a ring of public keys pk_{ring}. The $issue$ reflects the context in which a particular vote is to be cast. Let $\lambda \in N$ be a security parameter which denotes the desired level of security. We can define Traceable Ring Signatures in technical terms as follows. A Traceable Ring Signature scheme consists of algorithms $< Gen,\ Sign,\ Verify,\ Trace >$ such that:

- Gen: is a probabilistic polynomial-time algorithm that takes a security parameter $\lambda \in N$ as input and outputs a public and secret key pair $(pk,\ sk)$.
- $Sign$: is a probabilistic polynomial-time algorithm that takes a secret key, sk_i, where $i \in N$, tag $\mathcal{L} = (issue,\ pk_{ring})$, and message $m \in \{0,1\}^*$ as input and outputs a signature σ.
- $Verify$: is a deterministic polynomial-time algorithm that takes tag $\mathcal{L} = (issue,\ pk_{\text{ring}})$, message $m \in \{0,1\}^*$, and signature σ as input and outputs a bit indicating the validity of the signature.

- $Trace$: is a deterministic polynomial-time algorithm that takes tag $\mathcal{L} = (issue,\ pk_{\text{ring}})$, and two message-signature pairs, (m, σ) , $\left(m^{'}, \sigma^{'}\right)$ as input and outputs string $\text{result} \in \{\text{indep},\ \text{linked},\ pk\},\ \text{where}\ pk \in pk_{ring}$ subject to the conditions implied by public traceability as defined in [12].

20.3.4 Merkle Patricia Trie

The Merkle Patricia Trie (MPT) is a distributed data structure that maintains a consistent and efficient key-value database [13]. It combines Merkle Tree [14] and Patricia Trie [15]. The Merkle Trees guarantee data integrity and verification using cryptographic hashing, while the Patricia Tries have an inherent structure to support efficient storage and retrieval properties. The data can be accessed through the node path traversal. A generic MPT structure has three node types: leaf, branch, and extension. One of the main features of MPT is the easy verification of state changes. The root of the MPT can be used to ensure that a particular value is consistent throughout the various distributed instances of a key-value database. This property is supported by the standard security guarantees of a cryptographic hash function [16]. Currently, MPT is employed in various blockchains such as Ethereum, Quorum, and many more, for storage tracking state changes of blockchain global state (in the form of a state trie), transactional data, and similar associated data [13].

20.4 Proposed Model

20.4.1 Notation

The notation throughout the text follows an indexed superscript for a parameter to form an association with network participants. In the subsequent sections, we will use TA to denote a KGC throughout our discussion. The notation $E^{(i)}$ and $TA^{(j)}$ denote Entity with index i and TA with index j, respectively. $Enc_s\,(m,\ k)$ and $Dec_s\,(c,\ k)$ denote symmetric key encryption of message m with key k and symmetric key decryption of ciphertext c with key k respectively. The state trie is denoted by $\mathcal{ST}$. The existence of an element e in the $\mathcal{ST}$ is denoted by $e \in \mathcal{ST}$, and consequently, the non-existence is denoted by $e \notin \mathcal{ST}$. The inclusion of an element e within $\mathcal{ST}$ is denoted by $\mathcal{ST} \leftarrow \mathcal{ST} \bigcup \{e\}$. Every network participant can query $\mathcal{ST}$ and can check for the existence of e in $\mathcal{O}\,(1)$ time through a query to a subset of validators. The

validators or authorized participants can also include an element within the state trie through consensus. The validators also have an internal mechanism $Disincentivize\,(\mathcal{S})$, which takes a set $\mathcal{S}$, and disincentivizes the public keys in the set $\mathcal{S}$ through consensus amongst the validators. The messages within the network participants are denoted as $\langle type, param1, param2 \ldots \rangle$ where the first parameter is always message type. The usage of other parameters will be highlighted in the respective algorithms. The methods $broadcast\,(msg)$ and $send\,(dest, msg)$ are used for message transmission. The broadcast sends the message msg to every validator in the network while the method $send$, sends the message msg to a network participant denoted by $dest$. For instance, $send\left(TA^{(j)},\ \langle\,\text{init}\,\rangle\right)$ denotes a message with type $init$ to $TA^{(j)}$. The source of this message depends on the context of other parameters or where it is mentioned in the text. The process of receiving a message is denoted by $on{-}recv\,(< message >)$. The operations regarding Traceable Ring Signatures are denoted by a subscript trs. The operations $Gen_{trs}(\lambda)$, $Sign_{trs}(sk_{trs}^{(i)},\ \mathcal{L},\ m)$, $Verify_{trs}(\mathcal{L},\ m, \sigma)$, $Trace_{trs}\left(\mathcal{L},\ (m, \sigma), (m^{'}, \sigma^{'})\right)$ are defined as in Subsection 1.3.3.

There exists a method called $lookup\left(P_0^{(j)}\right)$ which takes $P_0^{(j)}$ associated with $TA^{(j)}$ and returns corresponding $pk_{trs}^{(j)}$. The lookup table is built up during the network bootstrapping, as discussed in Subsection 1.4.3.1. The parameters associated with the network participants are mentioned in Table 20.1.

20.4.2 Network Architecture

The network architecture is assumed to take the form depicted in Figure 20.1. Two major network participants are Trusted Authorities (TA) and Entities (E). Formally we can state that the set of TAs is denoted by $TA = \{TA^{(1)}, TA^{(2)}, \ldots, TA^{(w)}\}$ and set of entities is denoted by $E = \{E^{(1)},\ E^{(2)}, \ldots, E^{(x)}\}$. It is also assumed that $|TA| = w$ and $|E| = x$. The network participants have communication links between each other where they can exchange data. The role of validators mentioned in Subsection 1.4 is delegated to the network participants of the set TA. It is also assumed that the majority of the validators are honest. Therefore, the network participants in set TA have the responsibility of updating and maintaining $\mathcal{ST}$ and executing $Disincentivize\,(\mathcal{S})$ consistently. $\mathcal{ST}$ is assumed to remain consistent for all operations. All the network participants can query $\mathcal{ST}$

Table 20.1 Notations.

Notation	Explanation
$TA^{(j)}$	trusted authority (TA) with index j
$E^{(i)}$	entity (E) with index i
$ID^{(i)}$	identifier for $E^{(i)}$
Z_q^*	the multiplicative group of integers modulo prime q
$s^{(i)}$	secret value of $E^{(i)}$
$s_{ta}^{(j)}$	secret value of $TA^{(j)}$
$P_0^{(j)}$	public parameter P_0 for $TA^{(j)}$
$Q^{(i)}$	intermediate value generated during partial key generation for entity $E^{(i)}$
$D^{(i)}$	partial private key for $E^{(i)}$
${D_{enc}}^{(i)}$	encrypted value of $D^{(i)}$
N	set of natural numbers
λ	a security parameter for desired security guarantees
$\sigma^{(i,j)}$	the TRS signature signed by $TA^{(j)}$ for key generation request from $E^{(i)}$
pk_{ring}	ring associated with TRS consisting of a fixed set of public keys
$\mathcal{L}^{(i)}$	tag value associated with the key generation for $E^{(i)}$

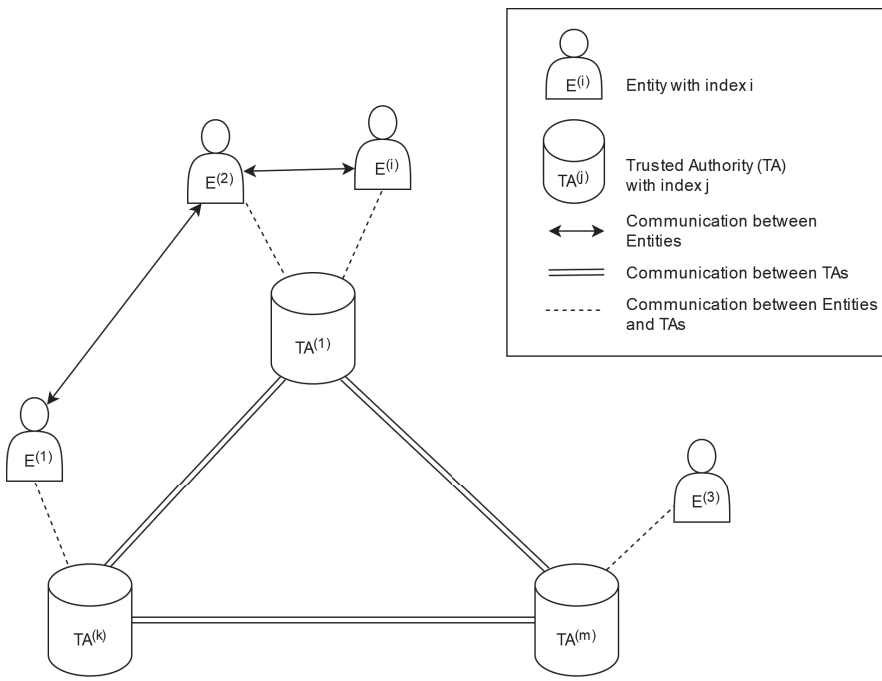

Figure 20.1 The underlying architecture consists of TA nodes and Entities requesting partial private key generation. In this model, any Entity node can start the protocol with a TA node of its choice.

in constant time by issuing a request to one or multiple $\mathcal{ST}$ maintainers. The network is assumed to be synchronous, and Δ_s bounds the network packet delivery time. The maximum time for the inclusion of an element in the $\mathcal{ST}$ and maximum processing time is bounded by Δ_a and Δ_p, respectively.

20.4.3 Protocol

This section discusses the mechanism for partial private and full private key generation. Firstly, we discuss the process of network bootstrapping and the protocol for private key generation between a TA and an Entity. In further discussion, we introduce algorithms for updating $\mathcal{ST}$, the $Audit$ Algorithm for enforcing accountability, and finally, the $collectAndTrace$ Algorithm, which is used in the detection of malicious TA (or TAs).

20.4.3.1 Network Bootstrapping

The process of network bootstrapping is presented in Figure 20.2. The TAs in the set TA all have the reference to public parameters $pp :=< G_{\mathbb{1}}, G_2, e, P, \mathcal{H}_1, \mathcal{H}_2 >$. $G_{\mathbb{1}}$ is an additive group and G_2 is a multiplicative group. Both $G_{\mathbb{1}}$ and G_2 are of prime order q. P is a generator of $G_{\mathbb{1}}$. The pairing e is an efficient Billinear Pairing such that $e : G_{\mathbb{1}} \times G_{\mathbb{1}} \rightarrow G_2$. The functions $\mathcal{H}_1$ *and* $\mathcal{H}_2$ are cryptographic hash functions such that $\mathcal{H}_1 : \{0,1\}^* \rightarrow G_{\mathbb{1}}$ and $\mathcal{H}_2 : \{0,1\}^* \rightarrow \{0,1\}^n$. It is important to note

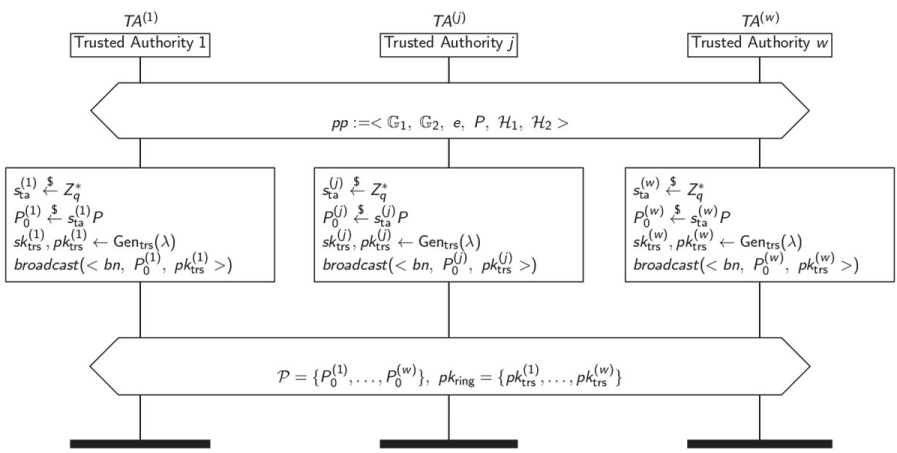

Figure 20.2 Network Bootstrapping.

that these hash functions are cryptographically secure with standard security guarantees.

Each $TA^{(j)}$ randomly samples its secret value $s_{ta}^{(j)}$ from Z_q^*. Then, it calculates its public key for the CL-PKC key generation mechanism $P_0^{(j)}$. After generating $P_0^{(j)}$, $TA^{(j)}$ runs $Gen_{trs}\,(\lambda)$ to obtain $sk_{trs}^{(j)}$ and $pk_{trs}^{(j)}$, which are its private and public keys for generating Traceable Ring Signatures. The node then broadcasts $P_0^{(j)}$ and $pk_{trs}^{(j)}$ with message type bn to all the other TAs in the set TA. Finally, all the TAs obtain set $\mathcal{P} = \{P_0^{(1)}, \ldots, P_0^{(w)}\}$ and the ring $pk_{ring} = \{pk_{trs}^{(1)}, \ldots, pk_{trs}^{(w)}\}$.

20.4.3.2 Key Generation

The flowchart in Figure 20.3 illustrates the logical sequence of the key generation process along with updating $\mathcal{ST}$ and executing of audit mechanism. The key generation procedure is illustrated in Figure 20.4. The private key generation is a procedure between two network participants, namely, $E^{(i)}$ with $ID^{(i)}$ and $TA^{(j)}$. Firstly, $E^{(i)}$ samples its secret value $s^{(i)}$ from Z_q^* and also generates the associated set of public keys $X^{(i)} \leftarrow s^{(i)} P_0^{(j)}$ and $Y^{(i)} \leftarrow s^{(i)} P$. After the generation of public keys $E^{(i)}$, sends an $init$ message containing $ID^{(i)}$, $X^{(i)}$ and $Y^{(i)}$. $TA^{(j)}$ on reception of $init$ message starts

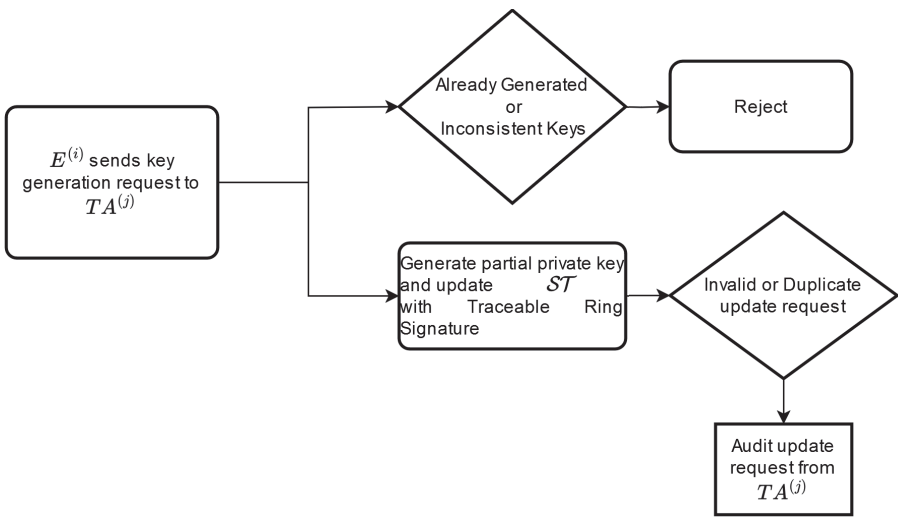

Figure 20.3 Flowchart for the full key generation and audit procedure.

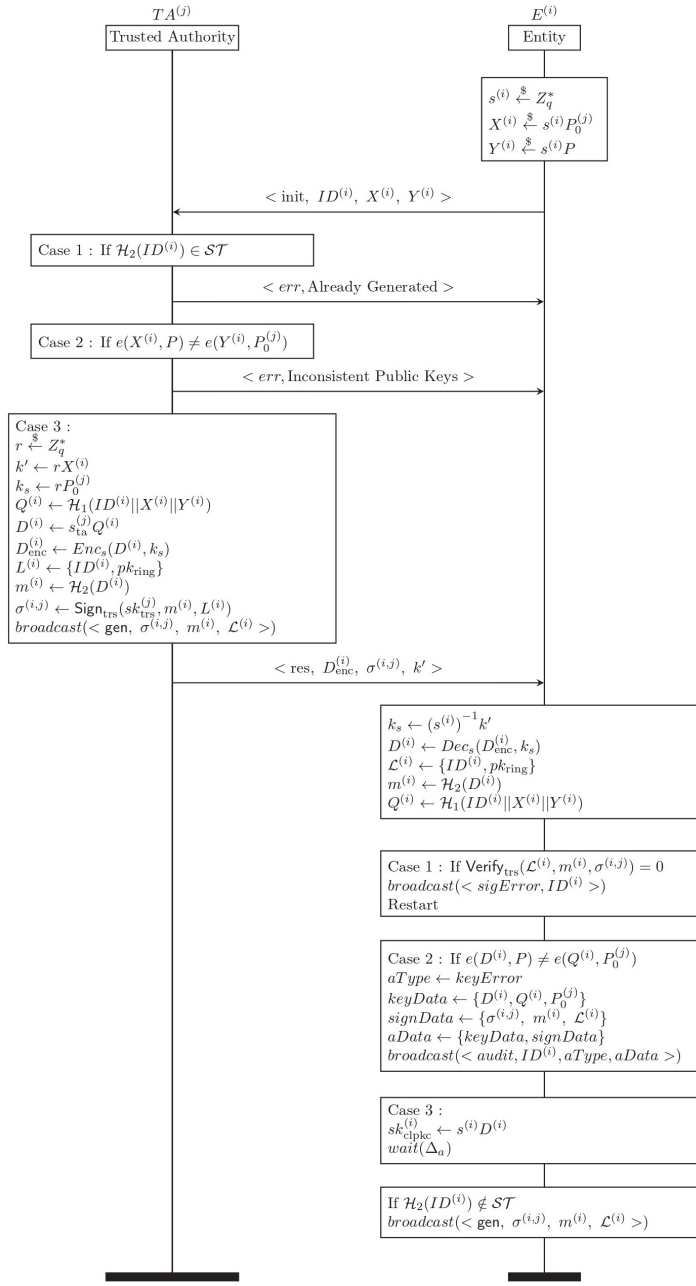

Figure 20.4 Key generation protocol.

the generation of partial private key $D^{(i)}$. Three scenarios can occur. Note that each case represents a stage in the process. Each case is a check which must be followed to reach the next one.

- **Case 1**: $D^{(i)}$ has already been generated for $ID^{(i)}$ as there exists an entry for $ID^{(i)}$ in the $\mathcal{ST}$. In this case, $TA^{(j)}$ aborts the partial key generation process and sends a message with type err stating that the key has already been generated.
- **Case 2**: If $e\left(X^{(i)}, P\right) \neq e\left(Y^{(i)}, P_0^{(j)}\right)$, it implies that the public keys generated are inconsistent or $P_0^{(j)}$was not used in for public key generation.
- **Case 3**: In this case, the public keys $X^{(i)}$and $Y^{(i)}$ are consistent and $TA^{(j)}$ can begin the partial private key generation process. It is important to note that the $D^{(i)}$ must be transmitted over an insecure channel. Therefore, it must remain confidential between $TA^{(j)}$and $E^{(i)}$. To this end, $TA^{(j)}$ samples r randomly from Z_q^* and calculates $k^{'} \leftarrow rX^{(i)}$. $k^{'}$ is the masked key for symmetric key encryption. Now, $TA^{(j)}$ calculates the actual key k_s for symmetric encryption by calculating $k_s \leftarrow rP_0^{(j)}$. In order to generate $D^{(i)}$, intermediate value $Q^{(i)}$ must be calculated as $Q^{(i)} \leftarrow \mathcal{H}_1(ID^{(i)} \mid\mid X^{(i)} \mid\mid Y^{(i)})$. $D^{(i)}$ is calculated as $D^{(i)} \leftarrow s_{ta}^{(j)} Q^{(i)}$. Next, $TA^{(j)}$ encrypts $D^{(i)}$ with key k_s using symmetric key encryption to obtain ${D_{enc}}^{(i)}$. In the subsequent step, $TA^{(j)}$ signs message $m^{(i)} \leftarrow \mathcal{H}_2\left(D^{(i)}\right)$ and tag $\mathcal{L}^{(i)} \leftarrow \{ID^{(i)}, \ pk_{ring}\}$ with $sk_{trs}^{(i)}$ to obtain $\sigma^{(i,j)}$. Finally, $TA^{(j)}$ broadcasts a message with type gen containing $< \ \sigma^{(i,j)}, m^{(i)}, \mathcal{L}^{(i)} \ >$ to every $TA \in TA$ (including itself) for updating $\mathcal{ST}$ and sends a message with type res to $E^{(i)}$ containing $< \ {D_{enc}}^{(i)}, \ \sigma^{(i,j)}, \ k^{'} \ >$.

20.4.3.3 Full Private Key Generation

On the reception of message of type res from $TA^{(j)}$, $E^{(i)}$ first tries to decrypt ${D_{enc}}^{(i)}$ to obtain $D^{(i)}$. For this it first obtains k_s by calculating $k_s \leftarrow \left(s^{(i)}\right)^{-1} k^{'}$. Using key k_s, it obtains $D^{(i)}$ by performing symmetric decryption operation as $D^{(i)} \leftarrow Dec_s\left(D_{enc}^{(i)}, k_s\right)$. Using the knowledge of $ID^{(i)}$ and pk_{ring}, it recovers the tag $\mathcal{L}^{(i)}$ as $\mathcal{L}^{(i)} \leftarrow \{ID^{(i)}, pk_{ring}\}$. The $m^{(i)}$ for traceable ring signature verification can be calculated as $m^{(i)} \leftarrow \mathcal{H}_2\left(D^{(i)}\right)$. Then $E^{(i)}$ calculates $Q^{(i)}$ as

$Q^{(i)} \leftarrow \mathcal{H}_1(ID^{(i)} \,||\, X^{(i)} \,||\, Y^{(i)})$. From this point on, there are three scenarios. Here, as well, note that each case represents a stage in the process. Each case is a check which must be followed to reach the next one.

- Case 1: If $Verify_{trs}\left(\mathcal{L}^{(i)}, m^{(i)}, \sigma^{(i,j)}\right) = 0$, the entity $E^{(i)}$ broadcasts the message $< sigError, ID^{(i)} >$ to stop the inclusion of $ID^{(i)}$ in $\mathcal{ST}$. It then restarts the key generation process. It is critical to note that in this case, we do not try to disincentivize the $TA^{(j)}$ as it is trivial to construct an invalid traceable ring signature. This might occur due to an adversary trying to block communication links for $E^{(i)}$ and sending an invalid signature. It is unfair if we try to disincentize $TA^{(j)}$ in such a case. Therefore, in this case, $E^{(i)}$ must restart the key generation procedure with the same or any other TA.
- Case 2: If $e\left(D^{(i)}, P\right) \neq e\left(Q^{(i)}, P_0^{(j)}\right)$ holds, it implies that the $TA^{(j)}$ did not calculate the value of $D^{(i)}$ consistently. The message signer can be held accountable since the traceable ring signature is valid. To this end, $E^{(i)}$ prepares the audit parameters $aType,\ keyData,\ signData,\ aData$ as follows:

$$aType \leftarrow keyError$$

$$keyData \leftarrow \{D^{(i)}, Q^{(i)}, P_0^{(j)}\}$$

$$signData \leftarrow \{\sigma^{(i,j)}, m^{(i)}, L^{(i)}\}$$

$$aData \leftarrow \{keyData,\ signData\ \}$$

  Finally, it broadcasts the message with type audit containing $< ID^{(i)},\ aType,\ aData >$ to all $TA \in TA$. The audit procedure by TA is discussed in Algorithm 2.
- Case 3: Finally, the full private key $sk_{\text{clpkc}}^{(i)}$ can be calculated by $E^{(i)}$ as $sk_{\text{clpkc}}^{(i)} \leftarrow s^{(i)} D^{(i)}$. For completeness we specify that $E^{(i)}$broadcasts the message with type gen containing $< sigma^{(i,j)},\ m^{(i)},\ \mathcal{L}^{(i)} >$ to all the $TA \in TA$. However, this can be optimized by checking if after Δ_a time, If $\mathcal{H}_2\left(ID^{(i)}\right) \in \mathcal{ST}$ is true or not, if not, then broadcast the message for the updating $\mathcal{ST}$.

20.4.3.4 Update $\mathcal{ST}$

During the key generation procedure between $TA^{(j)}$ and $E^{(i)}$, a $TA \in TA$, say $TA^{(k)}$ can receive multiple messages. These can be messages of type

gen, $sigError$, and audit. Updating $\mathcal{ST}$ concerns gen and $sigError$ message types. This process is described by Algorithm 1 . On reception of $< gen, \sigma^{(i,j)}, m^{(i)}, \mathcal{L}^{(i)} >$ in reference to $\mathcal{L}^{(i)}, TA^{(k)}$ first verifies the traceable ring signature. If the verification fails, it notifies $E^{(i)}$ to restart partial key generation and returns $\perp$. If $ID^{(i)}$ for which registration is done, exists already in $\mathcal{ST}$, $TA^{(k)}$ prepares to start the audit procedure as discussed in Algorithm 2. For this, initializes audit Type parameter $aType$ as $aType \leftarrow dupRegError$. It accumulates the received signature data in $signData$ and further includes it in audit data represented by $aData$. It then broadcasts the message with type audit attaching data $< ID^{(i)}, aType, aData >$. On the other hand, if $ID^{(i)} \notin \mathcal{ST}$, $TA^{(k)}$ waits for $2\Delta_s + \Delta_p$ time. If it does not receive message of type $sigError$ in reference to $ID^{(i)}$, it includes the $ID^{(i)}$ in the $\mathcal{ST}$, otherwise it notifies $E^{(i)}$ to restart partial key generation.

20.4.3.5 Audit Algorithm

In our protocol, we audit and disincentivize an entity for two cases, namely for $keyError$ and $dupRegError$, which correspond to inconsistent and duplicate key generation, respectively. The issuance of both cases has been discussed earlier. The audit of type $keyError$ is generated by an entity $E^{(i)}$ when the relation $e\left(D^{(i)}, P\right) = e\left(Q^{(i)}, P_0^{(j)}\right)$ does not hold. Similarly, the audit of type $dupRegError$ is initiated by a $TA^{(k)}$ when there is an attempt to generate partial keys for entity $E^{(i)}$ for which a key has already been generated. This can be an attempt to replace keys for entity $E^{(i)}$ or any other attack with similar intent. In the first step, $TA^{(k)}$ initializes an empty set $\mathcal{M}$ to store public keys of malicious TA (or TAs) for disincentivization. Next, it verifies whether the traceable ring signature in $aData$ is valid. If it is invalid, disincentivization cannot occur, and algorithm will return false results. This is because it is trivial to produce invalid signatures. To discuss the Audit Algorithm, we need to discuss the collectAndTrace as defined in Algorithm 3. It takes $ID^{(i)}, \mathcal{L}^{(i)}$ and $aData$. This Algorithm is executed as a subroutine in Algorithm 2. By the end of algorithm, $collectAndTrace$ returns set $\mathcal{M}$ containing the public key (or keys) of malicious TA (or TAs).

It is crucial to note that when it is called, it is executed in parallel, and its execution time is bounded by $\Delta_s + \Delta_p$. It first initializes the set $\mathcal{T}$ and the set $\mathcal{M}$. The set $\mathcal{T}$ stores received traceable ring signatures in form of a tuple $\left(m^{(l)}, \sigma^{(l)}\right)$. The set $\mathcal{M}$ will be used to add the public key (or keys) of malicious TA (or TAs). Next, it obtains m_{mal} and σ_{mal}, the suspected message and a valid traceable ring signature generated by

Algorithm 1: Update $\mathcal{ST}$ by $TA^{(k)}$

Data: First message $< gen,\ \sigma^{(i,j)},\ m^{(i)},\ \mathcal{L}^{(i)} >$ in reference to $\mathcal{L}^{(i)}$

Result: Update $\mathcal{ST}$ and execute Audit if necessary $on - recv(< gen,\ \sigma^{(i,j)},\ m^{(i)},\ \mathcal{L}^{(i)} >)$

if $Verify_{trs}\left(\mathcal{L}^{(i)}, m^{(i)}, \sigma^{(i,j)}\right) = 0$ **then**
  Failed! Notify $E^{(i)}$ to restart partial key generation
  return $\perp$
end
$id \leftarrow \mathcal{L}^{(i)}.issue$
if $\mathcal{H}_2(id) \in \mathcal{ST}$
  $aType \leftarrow dupRegError$
  $signData \leftarrow \{\sigma^{(i,j)}, m^{(i)}, \mathcal{L}^{(i)}\}$
  $aData \leftarrow \{signData\}$
  $broadcast(< audit, ID^{(i)}, aType, aData >)$
else
  $wait(\Delta_s + \Delta_p)$
  if not received $< sigError, ID^{(i)} >$ from $ID^{(i)}$ **then**
    $\mathcal{ST} \cup \{\mathcal{H}_2(\mathcal{L}^{(i)}.issue)\}$
  else
    Failed! Notify $E^{(i)}$ to restart partial key generation
  end
end

a malicious TA. It then starts a timer and waits for Δ_s time receiving traceable ring signatures generated by $TA \in TA$ for auditing in form of a message $< auditRes, ID^{(i)}, \mathcal{L}^{(i)}, m^{(l)}_{\text{audit}}, \sigma^{(l)}_{\text{audit}} >$. It verifies the $\mathcal{L}_{audit}{}^{(i)}$ and received tag $\mathcal{L}^{(i)}$ are same and the signature is valid. If both are true, then it includes the tuple $\left(m^{(l)},\ \sigma^{(l)}\right)$ in the set $\mathcal{T}$. It is important to note that this procedure of receiving and processing signatures also occurs in

parallel. Next, the $Trace_{trs}$ is used to find the malicious public key by running $Trace_{trs}\left(\mathcal{L}^{(i)},\ (m,\sigma)\,,\left(m_{\text{audit}}^{(l)},\sigma_{\text{audit}}^{(l)}\right)\right),\ \forall\ \left(m_{\text{audit}}^{(l)},\sigma_{\text{audit}}^{(l)}\right)\in\mathcal{T}$. If the $|\mathcal{T}|=w$, then we will have at least one $pk^{(y)}\in pk_{ring}$ in the set $\mathcal{M}$ with very high probability. If $|\mathcal{T}|<w$, algorithm includes all the non-signers. Lastly, if $|\mathcal{T}|>w$, few TAs signed the message more than once.

Thus, to get the signers who signed more than once, we run $Trace_{trs}$ for every pair of $\left(m_{\text{audit}}^{(l)},\sigma_{\text{audit}}^{(l)}\right)\in\mathcal{T}$ with every other such pair. In the end, we include such public keys in the set $\mathcal{M}$ and return $\mathcal{M}$. With the discussion of $collectAndTrace$, we can focus on the two cases for Audit Algorithm. Now, depending on the type of error, there are two possibilities:

Algorithm 1: Audit algorithm executed by $TA^{(k)}$

Data: First Audit Message $< audit, ID^{(i)}, aType, aData >$ in reference to $ID^{(i)}$

Result: $\top$ for successful disincentivization or $\perp$ for error

// Initialize empty set $\mathcal{M}$ for storing public keys of malicious TAs

$\mathcal{M} \leftarrow \phi$

$\sigma^{(i,j)} \leftarrow aData.signData.\sigma^{(i,j)}$

$\mathcal{L}^{(i)} \leftarrow aData.signData.\mathcal{L}^{(i)}$

$m^{(i)} \leftarrow aData.signData.m^{(i)}$

if $Verify_{trs}\left(\mathcal{L}^{(i)}, m^{(i)}, \sigma^{(i,j)}\right) = 0$ **then**

  return $\perp$

end

if $aType = keyError$

  if $e\left(D^{(i)}, P\right) \neq e\left(Q^{(i)}, P_0^{(j)}\right)$ **then**

    $\mathcal{L}_{audit}{}^{(i)} \leftarrow \{ID^{(i)}, pk_{ring}\}$

    $m_{\text{audit}}^{(k)} \leftarrow P_0^{(k)}$

    $\sigma_{audit}^{(k)} \leftarrow Sign_{trs}\left(sk_{trs}^{(k)}, m_{audit}^{(k)}, \mathcal{L}^{(i)}\right)$

    $broadcast\left(< auditRes, ID^{(i)}, \mathcal{L}^{(i)}, m_{audit}^{(k)}, \sigma_{audit}^{(k)} >\right)$

    $\mathcal{M} \leftarrow collectAndTrace\left(ID^{(i)}, \mathcal{L}_{audit}{}^{(i)}, aData\right)$

    $wait\left(\Delta_s + \Delta_p\right)$

$Disincentivize(\mathcal{M})$

end

end

if $aType = dupRegError$ **then**

$\mathcal{L}_{audit}{}^{(i)} \leftarrow \{ID^{(i)}, pk_{ring}\}$

$m_{\text{audit}}^{(k)} \leftarrow P_0^{(k)}$

$\sigma_{audit}^{(k)} \leftarrow Sign_{trs}\left(sk_{trs}^{(k)}, m_{audit}^{(k)}, \mathcal{L}^{(i)}\right)$

$broadcast\left(< auditRes, ID^{(i)}, \mathcal{L}^{(i)}, m_{audit}^{(k)}, \sigma_{audit}^{(k)} >\right)$

$\mathcal{M} \leftarrow collectAndTrace\left(ID^{(i)}, \mathcal{L}_{audit}{}^{(i)}, aData\right)$

$wait\left(\Delta_s + \Delta_p\right)$

$Disincentivize(\mathcal{M})$

end

return ⊤

- Case $keyError$: In the case of $keyError$, the validity of the claim that the generated key is invalid is verified. If the claim is true, $TA^{(k)}$ prepares a traceable ring signature for the audit process endorsing its public key $P_0^{(k)}$ as the message and tag $\mathcal{L}_{audit}{}^{(i)} \leftarrow \{ID^{(i)}, pk_{\text{ring}}\}$. This tag facilitates the finding of malicious TA. After this, $TA^{(k)}$ broadcasts the message of type $auditRes$ with contents $ID^{(i)}, L^{(i)}$ and $m_{\text{audit}}^{(k)}$ is broadcasted to every other $TA \in TA$. $TA^{(k)}$ calls the function $collectAndTrace$ with parameters $\mathcal{L}_{audit}{}^{(i)}$ and $aData$. The $collectAndTrace$ executes in parallel to return the set $\mathcal{M}$ in utmost $\Delta_s + \Delta_p$ time. Finally, the nodes disincentivize the participants in the set $\mathcal{M}$ by executing $Disincentivize\,(\mathcal{M})$.
- Case $dupRegError$: In this case as well, $TA^{(k)}$ prepares a traceable ring signature for the audit process endorsing its public key $P_0^{(k)}$ as the message and tag $\mathcal{L}_{audit}{}^{(i)} \leftarrow \{ID^{(i)}, pk_{ring}\}$ and then it broadcasts it with type $auditRes$. Finally, $\mathcal{M}$ is returned by executing $collectAndTrace$ in parallel in at most $\Delta_s + \Delta_p$ time, and the public keys in the set $\mathcal{M}$ are disincentivized.

Algorithm 3: $collectAndTrace$ algorithm executed by $TA^{(k)}$

Data: $ID^{(i)}, \mathcal{L}_{audit}{}^{(i)}, aData$

Result: Build and Return the set $\mathcal{M}$

\\ Initialize empty set M for storing malicious TRS

$\mathcal{M} \leftarrow \phi$

$m_{mal} \leftarrow aData.signData.m^{(i)}$

$\sigma_{mal} \leftarrow aData.signData.\sigma^{(i,j)}$

$timer \leftarrow 0$

while $timer \leq \Delta_s$

$on-recv\left(< auditRes, ID^{(i)}, \mathcal{L}^{(i)}, m_{audit}^{(l)}, \sigma_{audit}^{(l)} >\right)$

if $\mathcal{L}^{(i)} = \mathcal{L}_{audit}{}^{(i)} \wedge Verify_{trs}\left(\mathcal{L}^{(i)}, m_{audit}^{(l)}, \sigma_{audit}^{(l)}\right) = 1$ **then**

$\mathcal{T} \cup \{m_{audit}^{(l)}, \sigma_{audit}^{(l)}\}$

end

end

$\mathcal{M} \cup \{pk^{y}\} \mid \forall \{m_{audit}^{(y)}, \sigma_{audit}^{(y)}\} \in \mathcal{T} \wedge Trace_{trs}\left(\mathcal{L}^{(i)}, (m, \sigma), \left(m_{audit}^{(y)}, \sigma_{audit}^{(y)}\right)\right) = pk^{(y)}$

if $|\mathcal{T}| < w$

$signers \leftarrow \phi$

$signers \cup \{lookup(m_{audit}^{(s)}) \mid \exists \{m_{audit}^{(s)}, \sigma_{audit}^{(s)}\} \in \mathcal{T}$

$\mathcal{M} \cup \{pk_{ring} - signers\}$

end

if $|\mathcal{T}| > w$

$\mathcal{M} \cup \{ pk\hat{}\{(y)\} \} \mid \{ \forall \{m_{audit1}^{(y)}, \sigma_{audit1}^{(y)}\}, \{m_{audit2}^{(y)}, \sigma_{audit2}^{(y)}\} \in \mathcal{T} \wedge$

$Trace_{trs}\left(\mathcal{L}^{(i)}, \left(m_{audit1}^{(y)}, \sigma_{audit1}^{(y)}\right), \left(m_{audit2}^{(y)}, \sigma_{audit2}^{(y)}\right)\right) = pk^{(y)}$

end

return $\mathcal{M}$

This concludes the discussion of the established protocols and associated algorithms.

20.5 Empirical Results and Analysis

To evaluate the protocol, we designed a simulation using Golang [17]. For communication, protocol buffers were used. We implemented the Fujisaki-Suzuki Traceable Ring Signatures using the Ristretto prime-order group [18]. The group operations were done using the PBC library [19] to implement the original protocol. All the simulations were done on a Laptop with Intel(R) Core (TM) Ultra 7 165H 1.40 GHz processor with 32 GB Memory.

Since the generation of ring signatures is a bottleneck, we evaluated our implementation of traceable ring signatures sequentially and in parallel. As depicted in Figure 20.5, we evaluated the signature generation and verification process with ring sizes 10, 50, and 100. The time required to generate and verify 1000 signatures was recorded. As evident in Figure 20.5 (c), for a ring size of 100, the generation and verification took about 20 s. However, this is resolved when multiple signature generation and verifications are done in parallel; for a similar ring size of 100, the duration was reduced to 2.2 s. A similar reduction by a factor of 10 was also observed for ring sizes 10 and 50. Our evaluation shows that the scheme can be used for practical purposes.

Moreover, we simulated different instances with variations in the number of TAs and Entities. The results of the simulation are presented in Figure 20.6. The packet delays followed the Poisson distribution. We varied the number of Entities $(nEntities)$ by 50, 100, and 200. The number of Trusted Authorities (nTA) varied by 5, 10, and 20. For $nEntities = 50$, we observe that the

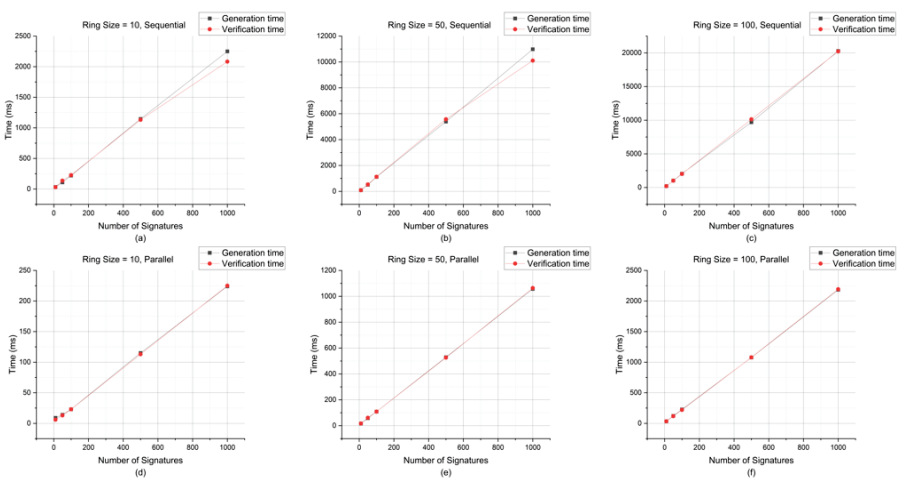

Figure 20.5 Benchmark for traceable ring signatures for sequential and parallel execution.

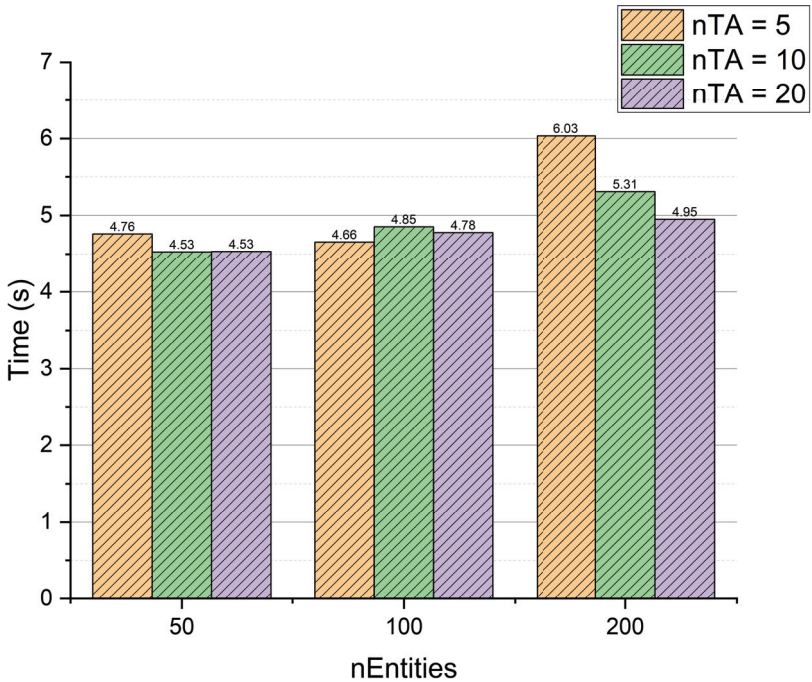

Figure 20.6 Key generation benchmarking for $\boldsymbol{nTA} = 5, 10, 20$ with different number of Entities.

total time for both $nTA = 10$ and $nTA = 20$ is upper bound by 4.5 s. This result is in line with the expectation that as more TAs are present, the load of key generation is distributed equitably. However, it is essential to notice that the size of the response from TA to Entity will be larger due to the increased size of the ring signature. This results in transmission and processing delays. The effect of increased signature sizes can be seen from the case for $nEntities = 100$. In this case, the configuration with $nTA = 5$ outperforms the configuration for $nTA = 10$ and $nTA = 20$. It is also interesting to observe that the configuration with $nTA = 20$ outperforms the configuration with $nTA = 10$. However, as more entities are added to the system, the effect of this decentralization is evident. The time required for key generation for $nEntities = 200$ is the minimum for the configuration with $nTA = 20$. Therefore, as more entities are added to the system, the protocol performs better with more trusted authorities. However, the requirement for the number of TAs varies from case to case.

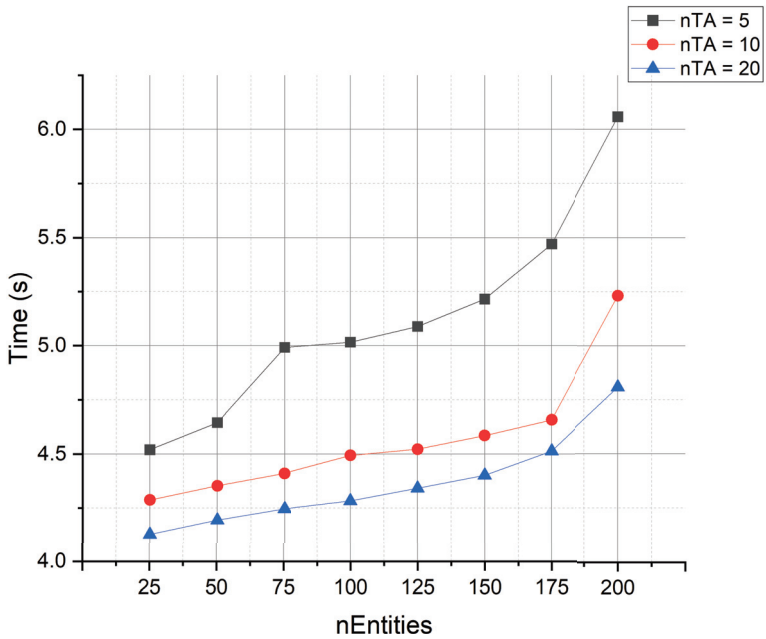

Figure 20.7 Single run comparison of key generation for $\boldsymbol{nTA} = 5, 10, 20$ with different number of Entities.

We also evaluate a single protocol run with various configurations with $nTA = 5$, 10, and 20, respectively. The time was recorded as more entities joined the system and requested a partial key generation and the corresponding data is reflected in Figure 20.7. Finally, the duration for successful completion of the complete protocol was recorded as well. The results are depicted in Figure 20.6. In this specific case, it can be inferred that the performance of the configurations $nTA = 10$ and $nTA = 20$ are very close to each other until $nEntities = 175$. As more entities are introduced, configuration with $nTA = 20$ achieves better time performance. However, as stated previously, the performance may vary on average due to large ring signatures. This can be optimized by fixing a specific ring for every key generation response.

20.6 Conclusions and Future Works

In this work, we have established a protocol to generate a CL-PKC-based private key with the network model and multiple trusted authorities. We

have also evaluated the computation and communication bottlenecks for efficient protocol implementation. The results indicate that the model can be adapted to accommodate the distributed nature of various applications. Considering the applications, the network model can have multiple use cases. For instance, in Federated Learning, a sensor node may not want to opt for a specific Trusted Authority confined to a geographic location or organization. This model facilitates the decentralization of the trust model in this case. It is important to note that we have established the accountability model for the general case of certificateless partial key generation. This work can be extended to accommodate the multiple trusted authorities in the existing models for Distributed Learning. Many of these models have established privacy-preserving solid models. However, inclusion must be made so the pre-existing guarantees are intact.

Acknowledgements

This work is supported by the European Network of Excellence dAIEDGE under Grant Agreement Nr. 101120726 and by CyberSecurity Research Flanders with reference number VR20192203.

References

[1] A. Shamir, "Identity-Based Cryptosystems and Signature Schemes," Advances in Cryptology, pp. 47-53, 1985. W442W7307

[2] D. Boneh and M. K. Franklin, "Identity-Based Encryption from the Weil Pairing," Proceedings of the 21st Annual International Cryptology Conference on Advances in Cryptology, pp. 213-229, 2001. W442W7307

[3] A.-R. S. S., "Certificateless Public Key Cryptography," Advances in Cryptology - ASIACRYPT 2003, pp. 452-473, 2003. W442W7307

[4] L. Zhang, Q. Wu, J. Domingo-Ferrer, B. Qin and P. Zeng, "Signatures in hierarchical certificateless cryptography: efficient constructions and provable security," Information Sciences, vol. 272, pp. 223-237, 2014. W442W7307

[5] L. Zhang, Q. Wu, J. Domingo-Ferrer and B. Qin, "Hierarchical certificateless signatures," 2010 IEEE/IFIP International Conference on Embedded and Ubiquitous Computing, pp. 572-577, 2010. W442W7307

[6] Y. Jiang, K. Zhang, Y. Qian and L. Zhou, "Anonymous and efficient authentication scheme for privacy-preserving distributed learning," IEEE Transactions on Information Forensics and Security, vol. 17, pp. 2227-2240, 2022. W442W7307

[7] Y. Ma, Q. Cheng and X. Luo, "2PCLA: Provable Secure and Privacy Preserving Enhanced Certificateless Authentication Scheme for Distributed Learning," IEEE Transactions on Information Forensics and Security, 2023. W442W7307

[8] X. Yuan, J. Liu, B. Wang, W. Wang, T. Li, X. Ma and W. Pedrycz, "Fedcomm: A privacy-enhanced and efficient authentication protocol for federated learning in vehicular ad-hoc networks," IEEE Transactions on Information Forensics and Security, 2023. W442W7307

[9] L. Zhang, J. Liu and R. Sun, "An efficient and lightweight certificateless authentication protocol for wireless body area networks," 2013 5th International Conference on Intelligent Networking and Collaborative Systems, pp. 637-639, 2013. W442W7307

[10] R. L. Rivest, A. Shamir and Y. Tauman, "How to leak a secret," Advances in Cryptology—ASIACRYPT 2001: 7th International Conference on the Theory and Application of Cryptology and Information Security Gold Coast, Australia, December 9–13, 2001 Proceedings 7, pp. 552-565, 2001. W442W7307

[11] M. Möser, K. Soska, E. Heilman, K. Lee, H. Heffan, S. Srivastava, K. Hogan, J. Hennessey, A. Miller, A. Narayanan and others, "An empirical analysis of traceability in the monero blockchain," arXiv preprint arXiv:1704.04299, 2017. W442W7307

[12] E. Fujisaki and K. Suzuki, "Traceable ring signature," International Workshop on Public Key Cryptography, pp. 181-200, 2007. W442W7307

[13] G. Wood and others, "Ethereum: A secure decentralised generalised transaction ledger," Ethereum project yellow paper, vol. 151, no. 2014, pp. 1-32, 2014. W442W7307

[14] R. C. Merkle, "Protocols for Public Key Cryptosystems," p. 122, 4 1980. W442W7307

[15] D. R. Morrison, "PATRICIA—Practical Algorithm To Retrieve Information Coded in Alphanumeric," Journal of the ACM (JACM), vol. 15, pp. 514 - 534, 1968. W442W7307

[16] W. Diffie and M. E. Hellman, "New Directions in Cryptography," Democratizing Cryptography: The Work of Whitfield Diffie and Martin Hellman, pp. 365-390, 2022. W442W7307

[17] R. Griesemer, R. Pike and K. Thompson, The Go Programming Language, 2009. W442W7307
[18] M. Hamburg, H. de Valence, I. Lovecruft and T. Arcieri, Ristretto: Prime-Order Elliptic Curve Groups, 2021. W442W7307
[19] B. Lynn, The PBC (Pairing-Based Cryptography) Library, 2006. W442W7307