Signatures in Hierarchical Certificateless Cryptography: Efficient Constructions and Provable Security [☆]

Lei Zhang[a], Qianhong Wu[b,c], Josep Domingo-Ferrer[b], Bo Qin[b,d]
Peng Zeng[a]

[a]*Shanghai Key Laboratory of Trustworthy Computing*
Software Engineering Institute
East China Normal University, Shanghai, China
[b]*UNESCO Chair in Data Privacy*
Department of Computer Engineering and Mathematics
Universitat Rovira i Virgili, Catalonia
[c]*School of Electronic and Information Engineering, Beihang University, China*
[d]*School of Information, Remin University of China, Beijing, China*
leizhang@sei.ecnu.edu.cn, {qianhong.wu,josep.domingo}@urv.cat, bo.qin@ruc.edu.cn,
pzeng@sei.ecnu.edu.cn

Abstract

Many efforts have been devoted in recent years to constructing secure schemes in certificateless cryptography. The aim is to eliminate the key escrow problem of identity-based cryptography. However, most of the work takes place in traditional certificateless cryptography, which suffers from the single point of failure problem. Hierarchical cryptography exploits a practical security model to mirror the organizational hierarchy in the real world and hence can eliminate the single point of failure problem. To incorporate the advantages of both types of cryptosystems, in this paper we instantiate hierarchical certificateless cryptography by formalizing the notion of hierarchical certificateless signatures. A concrete hierarchical certificateless signature scheme is also proposed. The security of our scheme is proven under the computational Diffie-Hellman assumption. As to efficiency, our scheme has constant complexity, regardless of the depth of the hierarchy. Therefore, our proposal is secure and scalable.

Keywords: Hierarchical cryptography; Certificateless cryptography; Hierarchical certificateless signature.

[☆]Parts of this paper appeared in [28].
[*]Corresponding author. Address: Software Engineering Institute, East China Normal University, Shanghai, China.

1. Introduction

Digital signatures are one of the most important primitives in public key cryptography. The public key of a signer can be leveraged by anyone to verify whether a signature attributed to this signer is valid or not. A problem for public key cryptography is that a user must be bound with her public key. The usual approach to solving this problem is to use a public key infrastructure (PKI), in which a trusted certificate authority authenticates the users' public keys. However, in practice, the management of public key certificates is extremely demanding in terms of computing time and storage space.

Identity-based cryptography (IBC) [24] was introduced to reduce the certificate management overhead in traditional PKI based cryptosystems. In IBC systems, the public key of a user is just her identity, e.g., an e-mail address, IP address, etc. Therefore, there is no need to maintain a complicated system to manage the addition, update or withdrawal of public key certificates. Hence, IBC systems are very efficient for some applications. Despite its advantages, the basic IBC system suffers from two limitations in practice. The first one is the single point of failure problem. In a basic IBC system, a single trusted third party (TTP) called Private Key Generator (PKG) is employed to generate a private key for each user by taking as input PKG's master secret key and the user's identity. Although having a single PKG would completely eliminate on-line lookup, it is undesirable for a large open network because the PKG may become a bottleneck. The PKG needs not only to generate private keys for a large number of users, but also to verify the identities of the users. The second limitation is the so-called key escrow problem. Since the PKG is used to generate the full private keys of the users in the system, the PKG knows each user's private key and a malicious PKG can forge signatures on behalf of any user without being detected. The above two limitations make IBC systems not scalable and not applicable to large open networks in which it is impractical to realize a third party fully trusted by all the distributed users.

Hierarchical identity-based cryptography (HIBC) [15] efficiently overcomes the single point of failure problem in IBC schemes. In HIBC, a *root* PKG is used to distribute the workload by delegating private key generation and identity authentication to *lower-level* PKGs. In this setting, multiple levels of PKGs and users as leaf nodes form a tree-like structure. HIBC was first instantiated by Horwitz and Lynn [15] with a hierarchical identity-based encryption (HIBE) scheme. Subsequent to the work in [15], Gentry and Silverberg [12] proposed a scalable HIBE scheme with full collusion resistance and chosen-ciphertext security. A hierarchical identity-based signature (HIBS) scheme was proposed in [12] as well. The concepts of HIBE and HIBS have been further investigated and more works can be found in [5, 7, 9, 20, 23]. HIBC efficiently overcomes the single-point bottleneck of IBC systems. However, it does not address the key escrow problem of such systems.

Certificateless cryptography (CLC) was introduced by Al-Riyami and Paterson to mitigate the key escrow problem. CLC also employs a third party called Key Generation Center (KGC) to help a user generate her private key. However, unlike the PKG in IBC, the KGC in CLC only extracts a partial private key for each user by taking as input a user's identity and the master key of the KGC. The full private key of a user is computed from the partial private key combined with some secret information chosen by the user herself. The corresponding public key is computed from the system's public parameters and the secret information of the user, and is finally published by the user herself. A number of contributions [1, 2, 10, 16, 17, 18, 19, 21, 26, 27, 30, 31, 29] have been devoted to the construction of secure schemes in CLC. In [1], Al-Riyami and Paterson proposed a certificateless encryption (CLE) scheme with chosen ciphertext security. A certificateless signature (CLS) scheme was also presented in [1] but no formal proof of security was provided. Subsequently, the security model of CLS schemes was formalized in [18] and the security of the Al-Riyami-Paterson CLS scheme was analyzed in this model. The security model of CLS schemes was further developed in [27] and later in [16, 17]. Among them, Huang et al. [17] revisited the security models of CLS schemes.

From the above discussions, one may find that neither CLC nor HIBC address the single point of failure and the secret key escrow problems in IBC systems *simultaneously*. To fill this gap, hierarchical certificateless cryptography (HCLC) is introduced in [1]. In an HCLC system, a *root* KGC distributes the workload by delegating partial private key generation and identity authentication to *lower-level* KGCs. Multiple levels of KGCs and users as leaf nodes form a tree-like structure. This hierarchical design matches the structure of social organizations in the real world very well. HCLC was first instantiated in [1] with a hierarchical certificateless encryption scheme. However, no security formalization has been provided for that scheme so far. This paper concentrates on hierarchical certificateless signatures (HCLSs), which inherit the advantages of hierarchical identity-based signatures without suffering from the key escrow problem.

1.1. Our Contributions

In the preliminary version [28] of this paper, we presented the first formalization of HCLSs as a new HCLC primitive. We explicitly defined the behaviors of adversaries against HCLSs with the help of a number of oracles. Two types of adversaries, i.e., Type I adversary and Type II adversary (see Section 2.2), were distinguished to simulate the collusive users and the malicious KGCs, respectively. Then we defined security as existential unforgeability against adaptive chosen-message attacks. We believe this strong security notion is suitable for most applications. We proposed a scalable HCLS scheme in which, to verify the validity of a signature from a signer, a verifier only needs to obtain the public parameters of the signer's *root* KGC, the user's identifying information (a user is uniquely identified by the path from the root to the user as a leaf node in the hierarchical tree) and the corresponding public key list. The verifier does not need an *on-line* lookup of the identities and public keys of the *lower-level* KGCs. The signing cost and the signature size are both constant, independent of the depth of the hierarchy. As for security, under the computational Diffie-Hellman (CDH) assumption, we showed that our scheme is provably secure against Type II adversaries under existential forgery attacks in the random oracle model [3].

In [28], we proposed a concrete HCLS scheme and gave a preliminary analysis of it. In this version, we provide a detailed analysis of the scheme. Firstly, we further study the behaviors of the adversaries against HCLSs. We show that the Type I and II adversaries in our security definition have similar abilities as the super Type I and Type II adversaries in [17] (see Section 2.2), who are the strongest adversaries against CLS so far. We also prove that our scheme is secure against enhanced Type I adversaries under existential forgery attacks. Secondly, we show how to detect a dishonest behavior of a malicious KGC in our scheme. Thirdly, we address the key revocation problem in HCLC. Fourthly, we discuss the computational cost of the proposed scheme. Finally, we distinguish our work from possible alternatives, namely a generic construction and a construction with short signatures.

The rest of this paper is organized as follows. In Section 2, we formalize the definition of HCLS schemes by defining the adversarial behaviors and the security notion of existential unforgeability. Our HCLS scheme is proposed in Section 3 and a detailed security analysis is given in Section 4. Section 5 distinguishes our work from other related HCLS constructions. We conclude our paper in Section 6.

2. Definitions

In this section, we formalize the notion of hierarchical certificateless signatures. We first define the algorithms underlying an HCLS scheme. Then we model the adversarial behaviors and present the main security notion, i.e., existential unforgeability against adaptive chosen-message attacks.

2.1. Hierarchical Certificateless Signatures

In an HCLS scheme, the entities include the *root* KGC, the *lower-level* KGCs and the users (signers). Each entity has a unique public identity ID. These entities form a structure of a hierarchical tree where the root KGC is at the top and the users are at the bottom, with the *lower-level* KGCs as the nodes between them.

In the hierarchy, each entity is uniquely identifiable by the path from the *root* KGC to the entity itself. Hence, we use an identity tuple $(ID_0, ..., ID_n)$ as the information to identify the *position* of an entity with ID_n in the hierarchy, where n is the tree depth from the *root* KGC with ID_0 to the entity with ID_n.

For $0 \leq i \leq n-1$ and $n \geq 1$, the entity with identity ID_i is the parent node of the entity with identity ID_{i+1}. The entity with identity ID_n can be either a *lower-level* KGC or a user as a leaf node[1]. For clarity, we use $\mathcal{K}_n$ to represent a *lower-level* KGC at level n, and $\mathcal{U}_n$ to represent a user at the same level, respectively. Specifically, $\mathcal{K}_0$ denotes the *root* KGC with identity ID_0, and if $n = 1$ for all the users, the system degenerates to a regular certificateless signature scheme.

An HCLS scheme consists of eight polynomial-time (probabilistic or deterministic) algorithms: Root-Setup, Lower-Level-Setup, Set-Secret-Value, Set-Public-Key, Partial-Private-Key-Extract, Set-Private-Key, Sign and Verify.

- Root-Setup: It is run by the *root* KGC $\mathcal{K}_0$. On input a security parameter λ, $\mathcal{K}_0$ selects a *root* private key s_0 and a list of public system parameters *params*. *params* will be a common input to the rest of algorithms. For simplicity, we omit it in the following specifications.

- Lower-Level-Setup: This is an interactive protocol that allows *lower-level* KGCs to be added to the system. It runs between $\mathcal{K}_n$ with identifying information $(ID_0, ..., ID_n)$, corresponding public key list

[1]To distinguish whether an entity is KGC or a user, we can adopt the convention that the identity of a KGC begins with the prefix *KGC* and the identity of a user with the prefix *user*.

$(P_0, ..., P_n)$ and its parent $\mathcal{K}_{n-1}$ with identifying information $(ID_0, ..., ID_{n-1})$, corresponding public key list $(P_0, ..., P_{n-1})$. When the protocol begins, $\mathcal{K}_n$ chooses a secret value and generates its public key; then $\mathcal{K}_n$ sends its identity and public key to $\mathcal{K}_{n-1}$. After receiving the information, $\mathcal{K}_{n-1}$ calculates the partial private key for $\mathcal{K}_n$ and sends it to $\mathcal{K}_n$. If this protocol ends without failing, $\mathcal{K}_n$ obtains its full private key.

- Set-Secret-Value: This algorithm is run by $\mathcal{U}_n$. On input $\mathcal{U}_n$'s identifying information $(ID_0, ..., ID_n)$ and an ordered public key list $(P_0, ..., P_{n-1})$, where $P_i, 1 \leq i \leq n-1$ is the public key of $\mathcal{K}_i$, this algorithm outputs a secret value s_n. This secret value will be used by $\mathcal{U}_n$ to generate her private key.

- Set-Public-Key: This algorithm is run by $\mathcal{U}_n$. On input $\mathcal{U}_n$'s secret value s_n, identifying information $(ID_0, ..., ID_n)$ and the corresponding public key list $(P_0, ..., P_{n-1})$, this algorithm generates $\mathcal{U}_n$'s public key P_n.

- Partial-Private-Key-Extract: A user $\mathcal{U}_n$ may request her parent $\mathcal{K}_{n-1}$ to run this algorithm to generate a partial private key for her. On input $\mathcal{U}_n$'s identifying information $(ID_0, ..., ID_n)$, the corresponding public key list $(P_0, ..., P_n)$ and $\mathcal{K}_{n-1}$'s private key, this algorithm generates a partial private key D_n for $\mathcal{U}_n$.

- Set-Private-Key: This algorithm is run by $\mathcal{U}_n$. On input $\mathcal{U}_n$'s secret value s_n, partial private key D_n, identifying information $(ID_0, ..., ID_n)$ and the corresponding public key list $(P_0, ..., P_n)$, this algorithm generates $\mathcal{U}_n$'s private key S_n.

- Sign: It is run by a user (signer) $\mathcal{U}_n$. It accepts a message m, $\mathcal{U}_n$'s private key S_n, identifying information $(ID_0, ..., ID_n)$ and the corresponding public key list $(P_0, ..., P_n)$, and generates an HCLS σ on message m under $(ID_0, ..., ID_n, P_0, ..., P_n)$.

- Verify: This algorithm is run by a verifier. It accepts a message m, a purported HCLS σ, the signer's identifying information $(ID_0, ..., ID_n)$ and the public key list $(P_0, ..., P_n)$, and outputs $true$ or $false$ to represent that σ is valid or not, respectively.

Note that the user's public key is required as an input of the Partial-Private-Key-Extract algorithm. In other words, when a user $\mathcal{U}_n$ requests a partial private key from her parent $\mathcal{K}_{n-1}$, $\mathcal{U}_n$ should first generate her public key. This is slightly different from the definition in [1]. This adaptation enables us to exploit the public key binding technique in [1]. With that technique, the security level of an HCLS scheme can achieve trust level 3 according to [11], the same security level enjoyed by the traditional PKI-based public key cryptography. The above requirement can also be removed. However, like the schemes in [1], the security level of the resulting HCLS scheme can only achieve trust level 2 according to [11].

2.2. Security Model

Before defining the security of HCLS schemes, we first review the adversaries in CLC. A secure CLS scheme should resist the attacks from two types of adversaries, known as 'Type I adversary' and 'Type II adversary'. In [1], these two types of adversaries are respectively defined as follows:

- Type I adversary is an attacker (colluding with users) who does not know the KGC's private key, but who can replace the public key of any user with a value of his choice.
- Type II adversary is an attacker (colluding with KGC) who owns the KGC's private key but cannot perform any public key replacement.

In [16, 17], Type II adversary is enhanced by additionally allowing him to replace the public key of any user except the target one. That is, the attacker can collude with the KGC and all users except the target user. We observe that this type of attack is also possible in a hierarchical setting. Hence, we respectively define the Type I adversary and the Type II adversary as follows:

- Type I adversary $\mathcal{A}_I$ does not know the *root* KGC's private key or the partial private key (or private key) of any *lower-level* KGC, but he can replace the public key of any entity except the root KGC with a value of his choice.
- Type II adversary $\mathcal{A}_{II}$ knows the private key of any KGC (including the *root* and all *lower-level* KGCs), and can replace the public key of any *lower-level* KGC and any user except $\mathcal{A}_{II}$'s target user.

The security model for HCLS schemes is defined by two games which involve a challenger and a Type I or II adversary. In the games, the challenger sets up the system parameter and controls several oracles defined below:

- Lower-Level-Setup Oracle: The input of this oracle is a KGC $\mathcal{K}_n$'s identifying information $(ID_0, ..., ID_n)$. It generates the *secret value* s_n and *public key* P_n for $\mathcal{K}_n$. We require that the public key of $\mathcal{K}_i$ has been already generated for $1 \leq i \leq n-1$. In other words, the attacker can only query $(ID_0\ , ..., ID_n)$ after querying $(ID_0, ..., ID_i)$ for $1 \leq i \leq n-1$.

- Reveal-KGC Oracle: The input of this oracle is a KGC $\mathcal{K}_n$'s identifying information $(ID_0, ..., ID_n)$, and the corresponding public key list $(P_0, ..., P_n)$. It outputs the partial private key D_n of $\mathcal{K}_n$.

- Public-Key Oracle: The input of this oracle is $\mathcal{U}_n$'s identifying information $(ID_0, ..., ID_n)$. It outputs the public key P_n of user $\mathcal{U}_n$.

- Partial-Private-Key Oracle: The input of this oracle is $\mathcal{U}_n$'s identifying information $(ID_0, ..., ID_n)$, and the corresponding public key list $(P_0, ..., P_n)$. It outputs the partial private key D_n of $\mathcal{U}_n$.

- Secret-Value Oracle: The input of this oracle is $\mathcal{U}_n$'s identifying information $(ID_0, ..., ID_n)$. It outputs the secret value s_n of $\mathcal{U}_n$. Note that if the public key of a user is replaced by the adversary, the challenger will not know the corresponding secret value. In this case, the challenger returns the original secret value of the user. The oracle service in this situation models the collusion between the attacker and the user.

- Public-Key-Replacement Oracle: On input $(ID_0, ..., ID_n, P'_n)$, this oracle sets P'_n as the new public key of the entity (either a user $\mathcal{U}_n$ or a KGC $\mathcal{K}_n$) whose identifying information is $(ID_0, ..., ID_n)$. Here P'_n is an element chosen from the public key space by the adversary.

- Sign Oracle: The input of this oracle is a tuple $(ID_0, ..., ID_n, P_0, ..., P_n, m)$. It outputs a valid HCLS σ on message m, where $(ID_0, ..., ID_n)$ is the identifying information of a user $\mathcal{U}_n$ and $(P_0, ..., P_n)$ is the corresponding public key list chosen by the adversary.

A Type I or II adversary is allowed to query some or all of the above oracles.

In [17], Type I/II adversary is further divided into three types, namely normal, strong and super Type I/II adversaries, depending on which kind of signatures an adversary can query during the Sign Oracle queries. A normal Type I/II adversary can only obtain some message-signature pairs which are valid under the original public key from the target signer. A strong adversary can obtain message-signature pairs which are valid under the replaced public key if he can supply the secret value corresponding to the replaced public key. Finally, a super adversary can obtain some message-signature pairs which are valid under the public key chosen by himself without supplying the secret value corresponding to the public key. From the above definitions, it is easy to see that the super Type I/II adversary is the strongest one.

Now we present the security model for HCLS schemes. The model is defined by the following two games played between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}_I$ or $\mathcal{A}_{II}$. In the games, we treat Type I and II adversaries as super Type I and II adversaries.

Game I (for *Type I Adversary*)

This game is played between a challenger $\mathcal{C}$ and a Type I adversary $\mathcal{A}_I$, and comprises three phases.

Phase I-1: $\mathcal{C}$ first runs Root-Setup to generate the *root* private key and the system parameter list *params*. Then $\mathcal{C}$ sends *params* to $\mathcal{A}_I$ but keeps the *root* private key.

Phase I-2: In this phase, $\mathcal{A}_I$ can adaptively query the Lower-Level-Setup Oracle, Public-Key Oracle, Partial-Private-Key Oracle, Secret-Value Oracle, Public-Key-Replacement Oracle and Sign Oracle defined above. However, before querying those oracles, we require that the public keys corresponding to the identities in the identifying information have been already generated. That is, the hierarchy of the *lower-level* KGCs has been fixed before the attacker queries the rest of oracles. This requirement is reasonable because, in practice, the hierarchy is formed before the users join the system or generate signatures.

Phase I-3: Finally, $\mathcal{A}_I$ outputs a tuple $(m^*, \sigma^*, ID_0^*, ..., ID_n^*, P_0^*, ..., P_n^*)$, in which $(ID_0^*, ..., ID_n^*)$ is the identifying information of the target user $\mathcal{U}_n^*$, $(P_0^*, ..., P_n^*)$ is the public key list chosen by $\mathcal{A}_I$, $ID_0^* = ID_0$ and $P_0^* = P_0$.

We say that $\mathcal{A}_I$ wins Game I if the above tuple satisfies the following requirements:

1. σ^* is a valid signature on m^* under $(ID_0^*, ..., ID_n^*, P_0^*, ..., P_n^*)$.
2. $(ID_0^*, ..., ID_n^*, P_0^*, ..., P_n^*)$ has never been submitted to the Partial-Private-Key Oracle.
3. $(ID_0^*, ..., ID_n^*, P_0^*, .., P_n^*, m^*)$ has never been submitted to the Sign Oracle.

We note that, in the above game, $\mathcal{A}_I$ is not allowed to query the Reveal-KGC Oracle. This restriction is reasonable, because $\mathcal{A}_I$'s final goal is to generate a signature on behalf of his target user. If he can query the Reveal-KGC Oracle, then he can learn the partial private key of a *lower-level* KGC. Since $\mathcal{A}_I$ can replace the public key of any entity, he can extract the full private key of the *lower-level* KGC corresponding to the replaced public key. Finally, $\mathcal{A}_I$ can forge any signature of any user that is valid under the revealed KGC's identifying information and the corresponding public keys, and the identity of the user and the corresponding public key.

Game II (for *Type II Adversary*)

This game is played between a challenger $\mathcal{C}$ and a Type II adversary $\mathcal{A}_{II}$. The game comprises three phases as follows.

Phase II-1: $\mathcal{C}$ runs Root-Setup to generate the *root* private key and the system parameter list *params*, then $\mathcal{C}$ sends *params* and *root* private key to $\mathcal{A}_{II}$.

Phase II-2: Similarly to Game I, $\mathcal{A}_{II}$ can query the following oracles as defined in Game I: Lower-Level-Setup Oracle, Public-Key Oracle, Partial-Private-Key Oracle, Secret-Value Oracle, Public-Key-Replacement Oracle and Sign Oracle. Furthermore, we additionally allow $\mathcal{A}_{II}$ to query the Reveal-KGC Oracle, which is used to reveal the partial private keys of *lower-level* KGCs.

Phase II-3: Finally, $\mathcal{A}_{II}$ outputs a tuple $(m^*, \sigma^*, ID_0^*, ..., ID_n^*, P_0^*, ..., P_n^*)$, in which $(ID_0^*, ..., ID_n^*)$ is the identifying information of the target user $\mathcal{U}_n^*$ and $(P_0^*, ..., P_n^*)$ is the corresponding public key list, $ID_0^* = ID_0$ and $P_0^* = P_0$.

We say that $\mathcal{A}_{II}$ wins Game II if the above tuple satisfies the following requirements:

1. σ^* is a valid signature on m^* under $(ID_0^*, ..., ID_n^*, P_0^*, ..., P_n^*)$.
2. P_n^* is the original public key of the target user $\mathcal{U}_n^*$. That is, $(ID_0^*, ..., ID_n^*, P_n')$ has never been submitted to the Public-Key-Replacement Oracle, where P_n' denotes any public key except P_n^*.
3. $\mathcal{A}_{II}$ has never requested the secret value of the user possessing public key P_n^*.
4. $(ID_0^*, ..., ID_n^*, P_0^*, ..., P_n^*, m^*)$ has never been submitted to the Sign Oracle .

Definition 1. *An HCLS scheme is existentially unforgeable under adaptive chosen-message attacks iff the probability of success of $\mathcal{A}_I$ in Game I and $\mathcal{A}_{II}$ in Game II is negligible.*

3. A Hierarchical Certificateless Signature Scheme

3.1. Bilinear Maps

Let $\mathbb{G}_1$ be an additive group of prime order q and $\mathbb{G}_2$ be a multiplicative group of the same order. Let $e : \mathbb{G}_1 \times \mathbb{G}_1 \longrightarrow \mathbb{G}_2$ be a bilinear map with the following properties:

1. Bilinearity: $e(aP, bQ) = e(P, Q)^{ab}$ for all $P, Q \in \mathbb{G}_1, a, b \in \mathbb{Z}_q^*$.
2. Non-degeneracy: There exists $P \in \mathbb{G}_1$ such that $e(P, P) \neq 1$.
3. Computability: There exists an efficient algorithm to compute $e(P, Q)$ for any $P, Q \in \mathbb{G}_1$.

The Weil pairing [6] on elliptic curves is an example of such a map.

We review the computational Diffle-Hellman (CDH) problem in $\mathbb{G}_1$. The CDH assumption states that any polynomial-time algorithm has only negligible success probability of solving the CDH problem. This assumption underlies our scheme.

Computational Diffle-Hellman (CDH) Problem in $\mathbb{G}_1$: Given a generator P of $\mathbb{G}_1$ and (aP, bP) for unknown $a, b \in \mathbb{Z}_q^*$, compute abP.

3.2. The Proposal

- Root-Setup: On input a security parameter λ, this algorithm runs as follows:

  1. Choose $q, \mathbb{G}_1, \mathbb{G}_2, e$ as defined in Section 3.1.
  2. Select a random generator $P \in \mathbb{G}_1$.
  3. Choose $s_0 \in \mathbb{Z}_q^*$ at random as the *root* private key and set $P_0 = s_0P$ as the public key of the root KGC $\mathcal{K}_0$.
  4. Choose cryptographic hash functions $H_1, ..., H_4 : \{0, 1\}^* \longrightarrow \mathbb{G}_1$.

  The system parameters are $params = (q, \mathbb{G}_1, \mathbb{G}_2, e, P, P_0, H_1, ..., H_4, ID_0, \mathcal{M})$, where ID_0 is the identity of $\mathcal{K}_0$ and $\mathcal{M} = \{0, 1\}^*$ is the message space.

- Lower-Level-Setup: For $n \geq 1$, let $\mathcal{K}_n$'s identifying information be $(ID_0, ..., ID_n)$ and the public key of $\mathcal{K}_i$ be $P_i, 1 \leq i \leq n-1$. When $\mathcal{K}_n$ joins the system, it runs this protocol with its parent $\mathcal{K}_{n-1}$ as follows:

  1. $\mathcal{K}_n$ does the following:
     (a) Choose a random $s_n \in \mathbb{Z}_q^*$ as the secret value and set $P_n = s_nP$ as its public key.
     (b) Send (ID_n, P_n) to $\mathcal{K}_{n-1}$.

2. Let $\mathcal{K}_{n-1}$ possess private key (s_{n-1}, D_{n-1}). When receiving (ID_n, P_n) from $\mathcal{K}_n$, $\mathcal{K}_{n-1}$ does as follows:
   (a) Compute $Q_n = H_1(ID_0, ..., ID_n, P_0, ..., P_n)$.
   (b) Compute $D_n = D_{n-1} + s_{n-1}Q_n$. Here, for consistency, we define $D_0 = 0$.
   (c) Send D_n to $\mathcal{K}_n$ through a secure channel.
3. $\mathcal{K}_n$ sets its private key as (s_n, D_n).

- Set-Secret-Value: On input $\mathcal{U}_n$'s identifying information $(ID_0, ..., ID_n)$ and the corresponding public key list $(P_0, ..., P_{n-1})$, $\mathcal{U}_n$ chooses $s_n \in \mathbb{Z}_q^*$ at random and outputs s_n as her secret value.

- Set-Public-Key: On input $\mathcal{U}_n$'s secret value s_n, identifying information $(ID_0, ..., ID_n)$ and the corresponding public key list $(P_0, ..., P_{n-1})$, $\mathcal{U}_n$ outputs $P_n = s_n P$ as her public key.

- Partial-Private-Key-Extract: On input $\mathcal{U}_n$'s identifying information $(ID_0, ..., ID_n)$ and the corresponding public key list $(P_0, ..., P_n)$, $\mathcal{K}_{n-1}$ computes the partial private key for $\mathcal{U}_n$ as follows:

  1. Compute $Q_n = H_1(ID_0, ..., ID_n, P_0, ..., P_n)$.
  2. Select $x' \in \mathbb{Z}_q^*$ at random and compute $R' = x'P$.
  3. Compute $E = H_2(ID_0, ..., ID_n, P_0, ..., P_n)$.
  4. Compute $D'_n = D_{n-1} + s_{n-1}Q_n + x'E$.
  5. Output the partial private key $D_n = (R', D'_n)$.

  $\mathcal{U}_n$ can verify the validity of the partial private key by checking

  $$e(D'_n, P) \stackrel{?}{=} e(R', E) \prod_{i=1}^{n} e(P_{i-1}, Q_i),$$

  where $Q_i = H_1(ID_0, ..., ID_i, P_0, ..., P_i), 1 \le i \le n$.

- Set-Private-Key: On input $\mathcal{U}_n$'s secret value s_n, partial private key D_n, identifying information $(ID_0, ..., ID_n)$ and the corresponding public key list $(P_0, ..., P_n)$, $\mathcal{U}_n$ sets $S_n = (s_n, D_n)$ as her private key.

- Sign: Let the signer $\mathcal{U}_n$ have identifying information $(ID_0, ..., ID_n)$, public keys $(P_0, ..., P_n)$ and private key $S_n = (s_n, D_n)$, where $D_n = (R', D'_n)$. To sign a message $m \in \mathcal{M}$, $\mathcal{U}_n$ uses the private key S_n and performs the following steps of which the first four can be pre-computed:

1. Compute $Q_i = H_1(ID_0, ..., ID_i, P_0, ..., P_i)$ for $1 \leq i \leq n$.
2. Compute $E = H_2(ID_0, ..., ID_n, P_0, ..., P_n)$.
3. Compute $R = R' + xP, W = D'_n + xE$ for a randomly chosen value $x \in \mathbb{Z}_q^*$.
4. Compute $U = yP$ for a randomly chosen value $y \in \mathbb{Z}_q^*$.
5. Compute $F = H_3(ID_0, ..., ID_n, P_0, ..., P_n, m)$, $T = H_4(ID_0, ..., ID_n, P_0, ..., P_n, m)$.
6. Compute $V = W + s_n F + yT$.
7. Output $\sigma = \{R, U, V\}$ as the HCLS on m.

- Verify: To verify an HCLS $\sigma = \{R, U, V\}$ on message m under $(ID_0, ..., ID_n, P_0, ..., P_n)$[2], the verifier performs the following steps:
  1. Compute $Q_i = H_1(ID_0, ..., ID_i, P_0, ..., P_i)$ for $1 \leq i \leq n$.
  2. Compute $E = H_2(ID_0, ..., ID_n, P_0, ..., P_n)$.
  3. Compute $F = H_3(ID_0, ..., ID_n, P_0, ..., P_n, m)$, $T = H_4(ID_0, ..., ID_n, P_0, ..., P_n, m)$.
  4. Check whether

$$e(V, P) \overset{?}{=} e(R, E)e(U, T)e(P_n, F) \prod_{i=1}^{n} e(P_{i-1}, Q_i) \tag{1}$$

  holds with equality. Output *true* if the equality holds; otherwise, output *false*.

The technique adopted in our scheme and most of the existing signature schemes in HIBC is similar to that of the sequential aggregate signature. The sequential aggregate signature is computed by letting each signer add her signature to an aggregate signature sequentially. The size of the final aggregate signature is expected to be independent of the number of signers. In existing sequential aggregate signature schemes, all signers use the same signature generation algorithm to generate signatures. However, existing efficient certificateless aggregate signature schemes require all the signers to agree on a common string in advance [32]. In an HCLS scheme, it seems impractical for all the KGCs and signers to agree on a same common string. Our technique is slightly different from the traditional one. In our scheme, a signer and her parent use specific signature generation algorithms to generate signatures and partial private keys, respectively.

[2]We notice that the identifying information and the corresponding public key list may need to be provided to a signature verifier along with the signature itself. However, the signature itself consists of only three group elements.

3.3. Implementation Concerns

In an HCLS scheme, if a (*root* or *lower-level*) KGC replaces a public key of a user and forges a signature of this user, then the KGC leaves evidence of his bad behavior, and such an action can be easily detected. In our HCLS scheme, the partial private key delivered to a user binds the user's identifying information and the corresponding public key list. This binding effectively restricts the user to using a single public key, since the user can now only compute one private key from the partial private key. If a KGC replaces a public key of a user and forges a signature of this user, then the KGC necessarily leaves evidence of his bad behavior; hence, such an action can be easily detected, as the KGC is the only entity having that capability. Note that this scenario is equivalent to a CA forging a certificate in a traditional PKI system: the existence of two valid certificates for a single identity would imply the CA behaved maliciously.

The efficiency of an HCLS scheme is mainly dominated by the Sign and Verify algorithms. In our scheme, the computational cost to generate a signature is constant. A signer has to compute 2 scalar multiplication operations in G_1, 2 MapToPoint hash operations and 2 addition operations in G_1, if pre-computation is considered. To verify a signature using the Verify algorithm, the computational cost has constant complexity which is $(n+4)t_e + (n+3)t_H + (n+2)t_{mul}$, where t_e is the time to compute a pairing operation, t_H is the time to compute a MapToPoint hash operation, and t_{mul} is the time to compute a multiplication operation in G_2. One may note that the pairing operation is the most time-consuming one. In our scheme, the Sign algorithm requires no pairing operation. As to the Verify algorithm, a verifier needs to compute $n+4$ (usually n is very small) pairing operations to verify the validity of a signature. However, as remarked in [8], when computing multiple pairings in a product, the cost incurred by adding each extra pairing is significantly less than the cost of the first pairing. The reason is that the sequence of doublings in Miller's algorithm [22] can be amortized over all the pairings in a given product, in a very similar way to the multi-exponentiation algorithm. According to [13], the cost of each additional pairing relative to the cost of one single pairing is between 0.1 and 0.6, depending on the pairing used. If we consider computing multiple pairings in a product, the time cost to verify a signature can be reduced to $((n+2)\rho + 2)t_e + (n+3)t_H + (n+2)t_{mul}$, where ρ is an acceleration factor. If the Tate Pairing is chosen, ρ is about 0.5.

Key revocation is also a concern that needs to be addressed for HCLS schemes to be deployable in the real world. In practice, users' public keys may become invalid after a period of use due to various reasons, e.g., if they were occasionally compromised, or the users forget the corresponding secret keys. Hence, we suggest users to renew their keys periodically. For instance, their public keys can contain information on their validity period. This is an easy method to realize revocation of public keys in HCLC systems. We note that a similar approach has been employed in IBC and HIBC systems [6]. One may also realize more versatile revocation mechanisms by modifying those identity revocation approaches in IBC [4] and HIBC systems [14].

4. Security Analysis

Theorem 1. *If there exists a Type I adversary who has an advantage ϵ in forging a signature of our HCLS scheme in an attack modeled by the Game I of Section 2.2, within a time span τ for a security parameter λ; and who can make at most q_{H_1} (resp. $q_{H_2}, q_{H_3}, q_{H_4}, q_L, q_{PK}, q_{PPK}$, q_{SK}, q_{PKR} and q_S) times H_1 (resp. H_2, H_3, H_4,* Lower-Level-Setup Oracle*,* Public-Key Oracle*,* Partial-Private-Key Oracle*,* Secret-Key Oracle*,* Public-Key-Replacement Oracle *and* Sign Oracle*) queries, then the CDH problem in $\mathbb{G}_1$ can be solved within time $\tau + \mathcal{O}(q_{H_1} + q_{H_2} + q_{H_3} + q_{H_4} + q_L + q_{PK} + \max\{n\} q_{PPK} + \max\{n\} q_S)\tau_{\mathbb{G}_1}$ and with probability*

$$\epsilon' \geq (1 - \frac{3}{q_{PPK} + q_S + 3})^{q_{PPK}+q_S} (\frac{3}{q_{PPK} + q_S + 3})^3 \epsilon,$$

where $\tau_{\mathbb{G}_1}$ is the time to compute a scalar multiplication in $\mathbb{G}_1$.

Proof. Let $\mathcal{C}$ be a CDH challenger, $\mathcal{A}_I$ be a Type I adversary who can break the proposed HCLS scheme with probability ϵ in time τ under an adaptive chosen-message attack. Suppose that $\mathcal{C}$ is given an instance (P, aP, bP) of the CDH problem in $\mathbb{G}_1$. We show how $\mathcal{C}$ can use $\mathcal{A}_I$ to solve the CDH problem, i.e., to compute abP. We notice that, in the following proof, essentially, the solution of the CDH problem is the partial private key of a KGC at level 1. During the simulation we will embed aP into P_0 and bP into Q_1^*, where $Q_1^* = H_1(ID_0^*, ID_1^*, P_0^*, P_1^*)$ and (ID_0^*, ID_1^*) is the identifying information of the KGC at level 1 and (P_0^*, P_1^*) is the corresponding public key list.

Phase I-1: Firstly, $\mathcal{C}$ sets $P_0 = aP$, $\mathsf{params} = (q, \mathbb{G}_1, \mathbb{G}_2, e, P, P_0, H_1, H_2, H_3, H_4, ID_0, \mathcal{M})$ in which the rest of parameters are generated as in the real scheme. Then $\mathcal{C}$ forwards *params* to $\mathcal{A}_I$. In the sequel, we treat H_1, H_2, H_3 and H_4 as random oracles which are controlled by $\mathcal{C}$. For all random oracles, if the same query has been queried before, then for consistency the same answer will be returned.

Phase I-2: $\mathcal{C}$ answers $\mathcal{A}_I$'s queries as follows:

H_1 Oracle queries: $\mathcal{C}$ maintains an initially empty list H_1^{list}. On input $(ID_0, ..., ID_n, P_0, ..., P_n)$ in which $n \geq 1$, $\mathcal{C}$ picks a random $\alpha_n \in \mathbb{Z}_q^*$ and simulates the random oracle H_1 as follows:

- If $n = 1$, flip a coin $coin_{H_1}^n \in \{0, 1\}$ that yields 1 with probability δ and 0 with probability $1 - \delta$.
  - If $coin_{H_1}^n = 0$, compute $Q_n = \alpha_n P$. Return Q_n as the answer and add $(ID_0, ID_n, P_0, P_n, \alpha_n, Q_n, coin_{H_1}^n)$ to H_1^{list}.
  - Else, compute $Q_n = \alpha_n bP$. Return Q_n as the answer and add $(ID_0, ID_n, P_0, P_n, \alpha_n, Q_n, coin_{H_1}^n)$ to H_1^{list}.
- Else if $n \neq 1$, set $coin_{H_1}^n = 0$, compute $Q_n = \alpha_n P$. Return Q_n as the answer and add $(ID_0, ...ID_n, P_0, ..., P_n, \alpha_n, Q_n, coin_{H_1}^n)$ to H_1^{list}.

H_2 Oracle queries: $\mathcal{C}$ keeps an initially empty list H_2^{list}. On input $(ID_0, ..., ID_n, P_0, ..., P_n)$, $\mathcal{C}$ first submits (ID_0, ID_1, P_0, P_1) to the H_1 Oracle and retrieves the tuple $(ID_0, ID_1, P_0, P_1, \alpha_1, Q_1, coin_{H_1}^1)$ in H_1^{list}. Then $\mathcal{C}$ does the following:

- If $coin_{H_1}^1 = 1$, randomly select $\beta_i \in \mathbb{Z}_q^*$, flip a coin $coin_{H_2} \in \{0, 1\}$ that yields 1 with probability δ and 0 with probability $1 - \delta$
  - If $coin_{H_2} = 0$, set $E = \beta aP$. Add $(ID_0, ..., ID_n, P_0, ..., P_n, \beta, E, coin_{H_2})$ to H_2^{list} and return E as the answer.
  - Else if $coin_{H_2} = 1$, compute $E = \beta P$. Add $(ID_0, ..., ID_n, P_0, ..., P_n, \beta, E, coin_{H_2})$ to H_2^{list} and return E as the answer.
- Else $coin_{H_1}^1 = 0$, set $coin_{H_2} = 0$. Randomly select $\beta \in \mathbb{Z}_q^*$, compute $E = \beta P$. Return E as the answer and add $(ID_0, ...ID_n, P_0, ..., P_n, \beta, E, coin_{H_2})$ to H_2^{list}.

H_3 Oracle queries: $\mathcal{C}$ keeps an initially empty list H_3^{list}. On input $(ID_0, ..., ID_n, P_0, ..., P_n, m)$, $\mathcal{C}$ selects a random $\gamma \in \mathbb{Z}_q^*$ and computes $F = \gamma P$. $\mathcal{C}$ adds $(ID_0, ..., ID_n, P_0, ..., P_n, m, \gamma, F)$ to H_3^{list} and responds with F.

H_4 Oracle queries: $\mathcal{C}$ keeps an initially empty list H_4^{list}. On input $(ID_0, ..., ID_n, P_0, ..., P_n, m)$, $\mathcal{C}$ first submits $(ID_0, ..., ID_n, P_0, ..., P_n)$ to the H_2 Oracle, and retrieves the tuple $(ID_0, ..., ID_n, P_0, ..., P_n, \beta, E, coin_{H_2})$ in H_2^{list}. Then $\mathcal{C}$ does the following:

- If $coin_{H_2} = 1$, randomly select $\pi \in \mathbb{Z}_q^*$, and flip a coin $coin_{H_4} \in \{0, 1\}$ that yields 1 with probability δ and 0 with probability $1 - \delta$
  - If $coin_{H_4} = 0$, set $T = \pi aP$. Add $(ID_0, ..., ID_n, P_0, ..., P_n, m, \pi, T, coin_{H_4})$ to H_4^{list} and respond with T.
  - Else if $coin_{H_4} = 1$, set $T = \pi P$. Add $(ID_0, ..., ID_n, P_0, ..., P_n, m, \pi, T, coin_{H_4})$ to H_4^{list} and return T as the answer.
- Else, set $coin_{H_4} = 0$. Select $\pi \in \mathbb{Z}_q^*$ at random and compute $T = \pi P$. Add $(ID_0, ..., ID_n, P_0, ..., P_n, m, \pi, T, coin_{H_4})$ to H_4^{list} and return T as the answer.

Lower-Level-Setup Oracle queries: $\mathcal{C}$ keeps an initially empty list KGC^{list}. On input a KGC's identifying information $(ID_0, ..., ID_n)$, if $(ID_0, ..., ID_n)$ already appears in KGC^{list} in a tuple $(ID_0, ..., ID_n, ...; s_0, ..., s_n, ...; P_0, ..., P_n, ...)$, $\mathcal{C}$ responds with (s_n, P_n) (here we define $s_0 = \phi$). Else $\mathcal{C}$ does the following:

- If $n = 1$, select $s_1 \in \mathbb{Z}_q^*$ and compute $P_1 = s_1 P$. Add $(ID_0, ID_1, s_0, s_1, P_0, P_1)$ to KGC^{list} and return (s_1, P_1) as the answer.
- Else, recover $(ID_0, ..., ID_{n-1}; s_0, ..., s_{n-1}; P_0, ..., P_{n-1})$ from KGC^{list}, and compute $P_n = s_n P$ for a randomly chosen $s_n \in \mathbb{Z}_q^*$. Remove $(ID_0, ..., ID_{n-1}; s_0, ..., s_{n-1}; P_0, ..., P_{n-1})$ from KGC^{list}. Add $(ID_0, ..., ID_n; s_0, ..., s_n; P_0, ..., P_n)$ to KGC^{list} and return (s_n, P_n) as the answer.

Public-Key Oracle queries: $\mathcal{C}$ maintains an initially empty list $User^{list}$. On input $(ID_0, ..., ID_n)$, if $(ID_0, ..., ID_n, P_n, s_n)$ exists in $User^{list}$, return P_n as the answer. Else, randomly select $s_n \in \mathbb{Z}_q^*$, compute $P_n = s_n P$, return P_n as the answer and add $(ID_0, ..., ID_n, P_n, s_n)$ to $User^{list}$.

Partial-Private-Key Oracle queries: On input $(ID_0, ..., ID_n, P_0, ..., P_n)$, $\mathcal{C}$ first does the following two steps:

1. For $1 \leq i \leq n$, submit $(ID_0, ..., ID_i, P_0, ..., P_i)$ to the H_1 Oracle to generate the tuple $(ID_0, ..., ID_i, P_0, ..., P_i, \alpha_i, Q_i, coin_{H_1}^i)$.
2. Submit $(ID_0, ..., ID_n, P_0, ..., P_n)$ to the H_2 Oracle to obtain the tuple $(ID_0, ..., ID_n, P_0, ..., P_n, \beta, E, coin_{H_2})$ in H_2^{list}.

Then $\mathcal{C}$ does the following:

- If $coin_{H_1}^1 = 0$, for a randomly chosen value $x \in \mathbb{Z}_q^*$, compute $R_n = xP$, $D_n = \sum_{i=1}^n \alpha_i P_{i-1} + xE$ and return (R_n, D_n) as the answer.

- Else if $coin^1_{H_1} = 1$ and $coin_{H_2} = 0$, randomly select $x \in \mathbb{Z}_q^*$, set $R_n = xP - \beta^{-1}\alpha_1 bP$, $D_n = \sum_{i=2}^n \alpha_i P_{i-1} + xE$, and return (R_n, D_n) as the answer.

- Else abort.

In the above setting, $\mathcal{A}_I$ can get the partial private key of any user except that of the target user at the target position with the target public keys.
Secret-Value Oracle queries: On input $(ID_0, ..., ID_n)$, $\mathcal{C}$ submits $(ID_0, ..., ID_n)$ to the Public-Key Oracle and recovers the tuple $(ID_0, ..., ID_n, P_n, s_n)$ from $User^{list}$. $\mathcal{C}$ returns s_n as the answer.

Public-Key-Replacement Oracle queries: On input $(ID_0, ..., ID_n, P'_n)$, this oracle runs as follows.

- If ID_n is the identity of a user, do the following:
  1. Submit $(ID_0, ..., ID_n)$ to the Public-Key Oracle.
  2. Recover $(ID_0, ..., ID_n, P_n, s_n)$ from $User^{list}$ and set $P_n = P'_n$.

- Else if ID_n is the identity of a KGC, do the following:
  1. Submit $(ID_0, ..., ID_n)$ to the Lower-Level-Setup Oracle.
  2. Recover from KGC^{list} all the tuples $(ID_0, ..., ID_n, ...; s_0, ..., s_n, ...; P_0, ..., P_n, ...)$ which contain $ID_0, ..., ID_n$. Set $P_n = P'_n$ in all the tuples.

Sign Oracle queries: On input $(ID_0, ..., ID_n, P_0, ..., P_n, m)$, $\mathcal{C}$ first does the following:

1. For $1 \leq i \leq n$, submit $(ID_0, ..., ID_i, P_0, ..., P_i)$ to the H_1 Oracle and recover $(ID_0, ..., ID_i, P_0, ..., P_i,\ \alpha_i, Q_i, coin^i_{H_1})$ from H_1^{list}.
2. Submit $(ID_0, ..., ID_n, P_0, ..., P_n)$ to the H_2 Oracle and recover $(ID_0, ..., ID_n, P_0, ..., P_n, \beta, E, coin_{H_2})$ from H_2^{list}.
3. Submit $(ID_0, ..., ID_n, P_0, ..., P_n, m)$ to both H_3 Oracle and H_4 Oracle, and recover $(ID_0, ..., ID_n, P_0, ..., P_n, m, \gamma, F)$ from H_3^{list}, $(ID_0, ..., ID_n, P_0, ..., P_n, m, \pi, T, coin_{H_4})$ from H_4^{list}.

Then $\mathcal{C}$ generates the signature by the following procedure:

- If $coin^1_{H_1} = 0$, generate the signature as follows:
  1. Randomly select $R, U \in \mathbb{G}_1$.
  2. Compute $V = \beta R + \pi U + \gamma P_n + \sum_{i=1}^n \alpha_i P_{i-1}$.
  3. Output (R, U, V).

We show that (R, U, V) is a valid signature on m under $(ID_0, ..., ID_n, P_0, ..., P_n)$. From the simulations, $E = \beta P$, $F = \gamma P$, $T = \pi P$, $Q_i = \alpha_i P$ for $1 \leq i \leq n$, we have that

$$e(V, P) = e(R, E)e(U, T)e(P_n, F)\prod_{i=1}^{n} e(P_{i-1}, Q_i).$$

Hence, (R, U, V) is a valid signature.

- Else if $coin^1_{H_1} = 1$ and $coin_{H_2} = 0$, generate the signature as follows:
  1. Randomly select $U \in \mathbb{G}_1$, $x \in \mathbb{Z}_q^*$.
  2. Set $R = xP - \beta^{-1}\alpha_1 bP$.
  3. Compute $V = xE + \pi U + \gamma P_n + \sum_{i=2}^{n} \alpha_i P_{i-1}$.
  4. Output (R, U, V). It can be shown to be a valid signature by direct verification.

  From the simulations, $E = \beta a P$, $F = \gamma P$, $T = \pi P$, $Q_1 = \alpha_1 bP$, $Q_i = \alpha_i P$ for $2 \leq i \leq n$, we have that

$$\begin{aligned}
& e(V, P) \\
= \; & e(xE + \pi U + \gamma P_n + \sum_{i=2}^{n} \alpha_i P_{i-1}, P) \\
= \; & e(xP, E)e(U, \pi P)e(P_n, \gamma P)e(P_0, Q_1)^{-1}e(P_0, Q_1)\prod_{i=2}^{n} e(P_{i-1}, \alpha_i P) \\
= \; & e(R + \beta^{-1}\alpha_1 bP, E)e(P_0, Q_1)^{-1}e(U, T)e(P_n, F)\prod_{i=1}^{n} e(P_{i-1}, Q_i) \\
= \; & e(R, E)e(U, T)e(P_n, F)\prod_{i=1}^{n} e(P_{i-1}, Q_i).
\end{aligned}$$

- Else if $coin^1_{H_1} = 1$, $coin_{H_2} = 1$ and $coin_{H_4} = 0$, do the following:
  1. Randomly select $R \in \mathbb{G}_1$, $y \in \mathbb{Z}_q^*$.
  2. Set $U = yP - \pi^{-1}\alpha_1 bP$.
  3. Compute $V = \beta R + yT + \gamma P_n + \sum_{i=2}^{n} \alpha_i P_{i-1}$.
  4. Output (R, U, V). It can be shown to be a valid signature by direct verification.

From the simulations, $E = \beta P$, $F = \gamma P$, $T = \pi a P$, $Q_1 = \alpha_1 b P$, $Q_i = \alpha_i P$ for $2 \leq i \leq n$, we have that

$$\begin{aligned}
& e(V, P) \\
= \quad & e(\beta R + yT + \gamma P_n + \sum_{i=2}^{n} \alpha_i P_{i-1}, P) \\
= \quad & e(R, \beta P) e(yP, T) e(P_n, \gamma P) e(P_0, Q_1)^{-1} e(P_0, Q_1) \prod_{i=2}^{n} e(P_{i-1}, \alpha_i P) \\
= \quad & e(R, E) e(U + \pi^{-1} \alpha_1 b P, T) e(P_0, Q_1)^{-1} e(P_n, F) \prod_{i=1}^{n} e(P_{i-1}, Q_i) \\
= \quad & e(R, E) e(U, T) e(P_n, F) \prod_{i=1}^{n} e(P_{i-1}, Q_i).
\end{aligned}$$

- Else abort.

Phase I-3: In this phase, $\mathcal{A}_I$ outputs a tuple $(m^*, \sigma^*, ID_0^*, ..., ID_n^*, P_0^*, ..., P_n^*)$, where $\sigma^* = (R^*, U^*, V^*)$, $ID_0^* = ID_0$ and $P_0^* = P_0$. It requires that σ^* be a valid signature on m^* under $(ID_0^*, ..., ID_n^*, P_0^*, ..., P_n^*)$.

$\mathcal{C}$ aborts if one of the following two events does not happen:

1. *Event 1*: For $1 \leq i \leq n$, $(ID_0^*, ..., ID_i^*, P_0^*, ..., P_i^*, \alpha_i^*, Q_i^*, coin_{H_1}^{i^*})$ is in H_1^{list}. The tuple $(ID_0^*, ..., ID_n^*, P_0^*, ..., P_n^*, \beta^*, E^*, coin_{H_2}^*)$ is in H_2^{list}, $(ID_0^*, ..., ID_n^*, P_0^*, ..., P_n^*, m^*, \gamma^*, F^*)$ is in H_3^{list}, and $(ID_0^*, ..., ID_n^*, P_0^*, ..., P_n^*, m^*, \pi^*, T^*, coin_{H_4}^*)$ is in H_4^{list}.
2. *Event 2*: $coin_{H_1}^{1^*} = coin_{H_2}^* = coin_{H_4}^* = 1$.

If both events happen, from our simulations, we have $P_0 = aP, Q_1^* = \alpha_1^* bP, E^* = \beta^* P, F^* = \gamma^* P, T^* = \pi^* P$ and for $2 \leq i \leq n$, $Q_i^* = \alpha_i^* P$. Since the public key P_0 of the root KGC is not allowed to be replaced, we can denote $P_0^* = P_0$. Note that

$$\begin{aligned}
& e(V^*, P) \\
= \quad & e(R^*, E^*)e(U^*, T^*)e(P_n^*, F^*)\prod_{i=1}^{n} e(P_{i-1}^*, Q_i^*) \\
= \quad & e(R^*, \beta^* P)e(U^*, \pi^* P)e(P_n^*, \gamma^* P)e(P_0, \alpha_1^* bP)\prod_{i=2}^{n} e(P_{i-1}^*, \alpha_i^* P) \\
= \quad & e(\beta^* R^*, P)e(\pi^* U^*, P)e(\gamma^* P_n^*, P)e(aP, \alpha_1^* bP)e(\sum_{i=2}^{n} \alpha_i^* P_{i-1}^*, P).
\end{aligned}$$

We have

$$abP = \alpha_1^{*-1}(V^* - (\beta^* R^* + \pi^* U^* + \gamma^* P_n^* + \sum_{i=2}^{n} \alpha_i^* P_{i-1}^*)),$$

which is the solution of the CDH problem.

To complete the proof, we shall show that $\mathcal{C}$ solves the given instance of the CDH problem with probability at least ϵ'. First, we analyze the three events needed for $\mathcal{C}$ to succeed:

- $\mathcal{E}1$: $\mathcal{C}$ does not abort as a result of any of $\mathcal{A}_I$'s Partial-Private-Key Oracle and Sign Oracle queries.
- $\mathcal{E}2$: σ^* is a valid signature on m^* under $(ID_0^*, ..., ID_n^*, P_0^*, ..., P_n^*)$.
- $\mathcal{E}3$: Both *Event 1* and *Event 2* happen.

$\mathcal{C}$ succeeds if all of these events happen. Hence, we have that

$$\Pr[\mathcal{C} \text{ succeeds}] = \Pr[\mathcal{E}1 \wedge \mathcal{E}2 \wedge \mathcal{E}3] = \Pr[\mathcal{E}1]\Pr[\mathcal{E}2|\mathcal{E}1]\Pr[\mathcal{E}3|\mathcal{E}1 \wedge \mathcal{E}2].$$

Claim 1. The probability that $\mathcal{C}$ does not abort as a result of $\mathcal{A}_I$'s Partial-Private-Key Oracle and Sign Oracle queries is at least $(1-\delta)^{q_{PPK}+q_S}$, and, formally, we have that $\Pr[\mathcal{E}1] \geq (1-\delta)^{q_{PPK}+q_S}$.

Proof. Define $\neg PPK_{abort}$ to be the event "$\mathcal{C}$ does not abort as a result of $\mathcal{A}_I$'s Partial-Private-Key Oracle queries" and $\neg S_{abort}$ be the event "$\mathcal{C}$ does not abort as a result of $\mathcal{A}_I$'s Sign Oracle queries".

For a Partial-Private-Key Oracle query on $(ID_0, ..., ID_n, P_0, ..., P_n)$, $\mathcal{C}$ will abort iff both $coin_{H_1}^1 = 1$ and $coin_{H_2} = 1$ happen. Since $\mathcal{A}_I$ can make at most q_{PPK} times Partial-Private-Key Oracle queries, we have

$$\Pr[\neg PPK_{abort}] \quad \geq \quad (1-\delta)^{q_{PPK}}$$

For a Sign Oracle query on $(ID_0, ..., ID_n, P_0, ..., P_n, m)$, $\mathcal{C}$ will abort iff $coin^1_{H_1} = coin_{H_2} = coin_{H_4} = 1$. Since $\mathcal{A}_I$ can make at most q_S times Sign Oracle queries, we have

$$\Pr[\neg S_{abort}] \quad \geq \quad (1-\delta)^{q_S}$$

Hence, we have $\Pr[\mathcal{E}1] \geq (1-\delta)^{q_{PPK}+q_S}$. □

Claim 2. $\Pr[\mathcal{E}2|\mathcal{E}1] \geq \epsilon$.

Proof. If Algorithm $\mathcal{C}$ does not abort as a result of $\mathcal{A}_I$'s Partial-Private-Key Oracle and Sign Oracle queries, then Algorithm $\mathcal{A}_I$'s view is identical to its view in the real attack. Note that $\mathcal{A}_I$ is an attacker which can break the proposed scheme with probability ϵ (in time τ). Hence, $\Pr[\mathcal{E}2|\mathcal{E}1] \geq \epsilon$. □

Claim 3. The probability that $\mathcal{C}$ does not abort after $\mathcal{A}_I$ outputting a valid forgery is at least δ^3. Formally, $\Pr[\mathcal{E}3|\mathcal{E}1 \wedge \mathcal{E}2] \geq \delta^3$.

Proof. If events $\mathcal{E}1$ and $\mathcal{E}2$ have occurred, $\mathcal{C}$ will abort unless *Event 1* or *Event 2* happen. If *Event* 1 happens, that means $\mathcal{C}$ generates a valid message-signature pair (m^*, σ^*) under $(ID_0^*, ..., ID_n^*, P_0^*, ..., P_n^*)$ without querying the corresponding hash functions. This probability is negligible. On the other hand, $\Pr[Event\ 2] = \Pr[coin^{1^*}_{H_1} = coin^*_{H_2} = coin^*_{H_4} = 1] \geq \delta^3$. Therefore, $\Pr[\mathcal{E}3|\mathcal{E}1 \wedge \mathcal{E}2] \geq \delta^3$. □

From Claims 1, 2 and 3, we have that

$$\epsilon' = \Pr[\mathcal{C} \text{ succeeds}] = \Pr[\mathcal{E}1 \wedge \mathcal{E}2 \wedge \mathcal{E}3] \geq (1-\delta)^{q_{PPK}+q_S}\delta^3\epsilon.$$

When $\delta = \frac{3}{q_{PPK}+q_S+3}$, $(1-\delta)^{q_{PPK}+q_S}\delta^3\epsilon$ has the maximum value. Hence, $\mathcal{C}$ can set $\delta = \frac{3}{q_{PPK}+q_S+3}$ to maximize the success probability

$$\epsilon' \geq (1 - \frac{3}{q_{PPK}+q_S+3})^{q_{PPK}+q_S}(\frac{3}{q_{PPK}+q_S+3})^3\epsilon.$$

This completes the proof. □

Theorem 2. *If there exists a Type II adversary who has an advantage ϵ in forging a signature of our HCLS scheme in an attack modeled by the Game II of Section 2.2, within a time span τ for a security parameter λ; and who can make at most q_{H_1} (resp. $q_{H_2}, q_{H_3}, q_{H_4}, q_L, q_{PK}, q_{PPK}$, q_{SK}, q_{RK}, q_{PKR} and q_S) times H_1 (resp. H_2, H_3, H_4,* Lower-Level-Setup Oracle*,* Public-Key Oracle*,* Partial-Private-Key Oracle*,* Secret-Key Oracle*,* Reveal-KGC Oracle*,* Public-Key-Replacement Oracle *and* Sign Oracle*) queries, then the CDH problem in*

$\mathbb{G}_1$ can be solved within time $\tau + \mathcal{O}(q_{H_1} + q_{H_2} + q_{H_3} + q_{H_4} + q_L + q_{PK} + \max\{n\}(q_{PPK} + q_{RK} + q_S))\tau_{\mathbb{G}_1}$ and with probability

$$\begin{aligned} \epsilon' \quad &\geq \quad (1 - \frac{2}{q_{SK} + q_{PKR} + q_S + 2})^{q_{SK}+q_{PKR}+q_S} \cdot \\ &\qquad (\frac{2}{q_{SK} + q_{PKR} + q_S + 2})^2 \epsilon, \end{aligned}$$

where $\tau_{\mathbb{G}_1}$ is the time to compute a scalar multiplication in $\mathbb{G}_1$.

Proof. Let $\mathcal{C}$ be a CDH challenger who is given an instance (P, aP, bP) of the CDH problem in $\mathbb{G}_1$ and wants to compute abP. Let $\mathcal{A}_{II}$ be a forger who can break the proposed HCLS scheme under an adaptive chosen-message attack. We show how $\mathcal{C}$ can use $\mathcal{A}_{II}$ to solve the CDH problem, i.e., to compute abP for unknown $a, b \in \mathbb{Z}_q^*$. We notice that, to forge a valid signature, $\mathcal{A}_{II}$ needs to know $x_n^* F^*$, where x_n^* is the secret value of the target user and $F^* = H_3(ID_0^*, ..., ID_n^*, P_0^*, ..., P_n^*, m^*)$. Hence, in the following proof, we will embed aP and bP into F^* and P_n^*, respectively.

Phase II-1: Firstly, $\mathcal{C}$ selects $s_0 \in \mathbb{Z}_q^*$, sets $P_0 = s_0 P$, then selects the system parameters $\mathsf{params} = (q, \mathbb{G}_1, \mathbb{G}_2, e, P, P_0, H_1, H_2, H_3, H_4, ID_0, \mathcal{M})$, and then gives s_0 and *params to* $\mathcal{A}_{II}$. In the sequel, we treat H_1, H_2, H_3 and H_4 as random oracles which are controlled by $\mathcal{C}$.

Phase II-2: $\mathcal{C}$ simulates the random oracles as well as the oracles defined in our Game II as follows:

H_1 Oracle queries: $\mathcal{C}$ maintains an initially empty list H_1^{list}. On input $(ID_0, ..., ID_n, P_0, ..., P_n)$, the same answer from H_1^{list} will be given if the request has been asked before. Otherwise, $\mathcal{C}$ picks a random $\alpha_n \in \mathbb{Z}_q^*$, computes $Q_n = \alpha_n P$, returns Q_n as the answer and adds $(ID_0, ..., ID_n, P_0, ..., P_n, \alpha_n, Q_n)$ to H_1^{list}.

H_2 Oracle queries: $\mathcal{C}$ keeps an initially empty list H_2^{list}. On input $(ID_0, ..., ID_n, P_0, ..., P_n)$, the same answer from H_2^{list} will be given if the request has been asked before. Otherwise, $\mathcal{C}$ randomly selects $\beta \in \mathbb{Z}_q^*$, sets $E = \beta P$, returns E as the answer and adds $(ID_0, ..., ID_n, P_0, ..., P_n, \beta, E)$ to H_2^{list}.

H_3 Oracle queries: $\mathcal{C}$ keeps an initially empty list H_3^{list}. On input $(ID_0, ..., ID_n, P_0, ..., P_n, m)$, the same answer from H_3^{list} will be given if the request has been asked before. Otherwise, $\mathcal{C}$ randomly selects $\gamma \in \mathbb{Z}_q^*$, flips a coin $coin_{H_3} \in \{0, 1\}$ that yields 1 with probability δ and 0 with probability $1 - \delta$. If $coin_{H_3} = 0$, $\mathcal{C}$ computes $F = \gamma P$; else sets $F = \gamma aP$. Finally, $\mathcal{C}$

adds $(ID_0, ...ID_n, P_0, ..., P_n, m, \pi, F, coin_{H_3})$ to H_3^{list} and returns F as the answer.

H_4 Oracle queries: $\mathcal{C}$ keeps an initially empty list H_4^{list}. On input $(ID_0, ..., ID_n, P_0, ..., P_n, m)$, the same answer from H_4^{list} will be given if the request has been asked before. Otherwise, $\mathcal{C}$ randomly selects $\pi \in \mathbb{Z}_q^*$ and computes $T = \pi P$. Finally, $\mathcal{C}$ adds $(ID_0, ..., ID_n, P_0, ..., P_n, m, \pi, T)$ to H_4^{list} and returns T as the answer.

Lower-Level-Setup Oracle queries: $\mathcal{C}$ keeps an initially empty list KGC^{list}. On input a KGC's identifying information $(ID_0, ..., ID_n)$, $\mathcal{C}$ does the following:

- If $(ID_0, ..., ID_n)$ already appears in KGC^{list} in a tuple $(ID_0, ..., ID_n, ...; s_0, ..., s_n, ...; P_0, ..., P_n, ...)$, respond with (s_n, P_n).
- Else if $n = 1$, select $s_1 \in \mathbb{Z}_q^*$, compute $P_1 = s_1 P$, add $(ID_0, ID_1; s_0, s_1; P_0, P_1)$ to KGC^{list} and return (s_1, P_1) as the answer.
- Else, recover $(ID_0, ..., ID_{n-1}; s_0, ..., s_{n-1}; P_0, ..., P_{n-1})$ from KGC^{list} at first, then randomly select $s_n \in \mathbb{Z}_q^*$, compute $P_n = s_n P$, add $(ID_0, ..., ID_n; s_0, ..., s_n; P_0, ..., P_n)$ to KGC^{list}, remove $(ID_0, ..., ID_{n-1}; s_0, ..., s_{n-1}; P_0, ..., P_{n-1})$ from KGC^{list} and respond with (s_n, P_n).

Public-Key Oracle queries: $\mathcal{C}$ keeps an initially empty list $User^{list}$. On input $(ID_0, ..., ID_n)$, $\mathcal{C}$ does the following:

- If $(ID_0, ..., ID_n, P_n, s_n, coin_{PK})$ is in $User^{list}$, return P_n as the answer.
- Else flip a coin $coin_{PK} \in \{0, 1\}$ that yields 1 with probability δ and 0 with probability $1 - \delta$, randomly select $s_n \in \mathbb{Z}_q^*$ and do the following:
  - If $coin_{PK} = 0$, compute $P_n = s_n P$ and add $(ID_0, ..., ID_n, P_n, s_n, coin_{PK})$ to $User^{list}$ and return P_n as the answer.
  - Else, set $P_n = s_n bP$, return P_n as the answer and add $(ID_0, ..., ID_n, P_n, s_n, coin_{PK})$ to $User^{list}$.

Partial-Private-Key Oracle queries: On input $(ID_0, ..., ID_n, P_0, ..., P_n)$, $\mathcal{C}$ does the following:

1. For $1 \leq i \leq n$, submit $(ID_0, ..., ID_i, P_0, ..., P_i)$ to the H_1 Oracle to generate the tuple $(ID_0, ...ID_i, P_0, ..., P_i, \alpha_i, Q_i)$.
2. Submit $(ID_0, ..., ID_n, P_0, ..., P_n)$ to the H_2 Oracle and later find the tuple $(ID_0, ..., ID_n, P_0, ..., P_n, \beta, E)$ in H_2^{list}.

3. Randomly select $x \in \mathbb{Z}_q^*$, set $R_n = xP$, $D_n = \sum_{i=1}^{n} \alpha_i P_{i-1} + xE$, and return (R_n, D_n) as the answer.

Secret-Value Oracle queries: On input $(ID_0, ..., ID_n)$, $\mathcal{C}$ first submits $(ID_0, ..., ID_n)$ to the Public-Key Oracle then recovers the tuple $(ID_0, ..., ID_n, P_n, s_n, coin_{PK})$ from $User^{list}$. If $coin_{PK} = 0$, $\mathcal{C}$ returns s_n as the answer; else $\mathcal{C}$ aborts.

Public-Key-Replacement Oracle queries: On input $(ID_0, ..., ID_n, P'_n)$, this oracle does the following:

- If ID_n is the identity of a user, do the following:
  1. Submit $(ID_0, ..., ID_n)$ to the Public-Key Oracle.
  2. Recover $(ID_0, ..., ID_n, P_n, s_n, coin_{PK})$ from $User^{list}$. If $coin_{PK} = 1$ and $P'_n \neq s_n P$, abort; otherwise, set $P_n = P'_n$.
- Else ID_n is the identity of a KGC, so do the following:
  1. Submit $(ID_0, ..., ID_n)$ to the Lower-Level-Setup Oracle .
  2. Recover from KGC^{list} all the tuples $(ID_0, ..., ID_n, ...; s_0, ..., s_n, ...; P_0, ..., P_n, ...)$ which contain $ID_0, ..., ID_n$. Set $P_n = P'_n$ in all the tuples.

Reveal-KGC Oracle: On input $(ID_0, ..., ID_n, P_0, ..., P_n)$, $\mathcal{C}$ does the following:

1. For $1 \leq i \leq n$, first submit $(ID_0, ..., ID_i, P_0, ..., P_i)$ to the H_1 Oracle, then recover $(ID_0, ..., ID_i, P_0, ..., P_i, \alpha_i, Q_i)$ from H_1^{list}.
2. Compute $D_n = \sum_{i=1}^{n} \alpha_i P_{i-1}$.
3. Output D_n.

Sign Oracle queries: On input $(ID_0, ..., ID_n, P_0, ..., P_n, m)$, $\mathcal{C}$ first does the following:

1. For $1 \leq i \leq n$, submit $(ID_0, ..., ID_i, P_0, ..., P_i)$ to the H_1 Oracle and recover $(ID_0, ..., ID_i, P_0, ..., P_i, \alpha_i, Q_i)$ from H_1^{list}.
2. Submit $(ID_0, ..., ID_n, P_0, ..., P_n)$ to the H_2 Oracle and recover $(ID_0, ..., ID_n, P_0, ..., P_n, \beta, E)$ from H_2^{list}.
3. Submit $(ID_0, ..., ID_n, P_0, ..., P_n, m)$ to both H_3 Oracle and H_4 Oracle, and recover $(ID_0, ..., ID_n, P_0, ..., P_n, m, \gamma, F, coin_{H_3})$ from H_3^{list}, $(ID_0, ..., ID_n, P_0, ..., P_n, m, \pi, T)$ from H_4^{list}.

$\mathcal{C}$ then generates the signature as follows:

- If $coin_{H_3} = 0$, generate the signature as follows:

1. Randomly select $R, U \in \mathbb{G}_1$.
2. Compute $V = \beta R + \pi U + \gamma P_n + \sum_{i=1}^{n} \alpha_i P_{i-1}$.
3. Output (R, U, V).

From the simulations, $E = \beta P$, $F = \gamma P$, $T = \pi P$, $Q_i = \alpha_i P$ for $1 \leq i \leq n$, we have that

$$\begin{aligned} & e(V, P) \\ = \quad & e(\beta R + \pi U + \gamma P_n + \sum_{i=1}^{n} \alpha_i P_{i-1}, P) \\ = \quad & e(R, \beta P) e(U, \pi P) e(P_n, \gamma P) \prod_{i=1}^{n} e(P_{i-1}, \alpha_i P) \\ = \quad & e(R, E) e(U, T) e(P_n, F) \prod_{i=1}^{n} e(P_{i-1}, Q_i). \end{aligned}$$

- Else abort.

Phase II-3: In this phase, $\mathcal{A}_{II}$ outputs a tuple $(m^*, \sigma^*, ID_0^*, ..., ID_n^*, P_0^*, ..., P_n^*)$, where $\sigma^* = (R^*, U^*, V^*)$. It requires that σ^* is a valid signature on m^* under $(ID_0^*, ..., ID_n^*, P_0^*, ..., P_n^*)$.

$\mathcal{C}$ aborts if any of the following events is not satisfied:

1. *Event 3*: For $1 \leq i \leq n$, $(ID_0^*, ..., ID_i^*, P_0^*, ..., P_i^*, \alpha_i^*, Q_i^*)$ is in H_1^{list}, $(ID_0^*, ..., ID_n^*, P_0^*, ..., P_n^*, \beta^*, E^*)$ is in H_2^{list}, $(ID_0^*, ..., ID_n^*, P_0^*, ..., P_n^*, m^*, \gamma^*, F^*, coin_{H_3}^*)$ is in H_3^{list}, and $(ID_0^*, ..., ID_n^*, P_0^*, ..., P_n^*, m^*, \pi^*, T^*)$ is in H_4^{list}.
2. *Event 4*: $coin_{H_3}^* = coin_{PK}^* = 1$, where $coin_{PK}^*$ is in the tuple $(ID_0^*, ..., ID_n^*, P_0^*, ..., P_n^*, m^*, \gamma^*, F^*, coin_{H_3}^*)$ in $User^{list}$.

If $\mathcal{C}$ does not abort, we have $E^* = \beta^* P, F^* = \gamma^* aP, T^* = \pi^* P$ and for $1 \leq i \leq n$, $Q_i^* = \alpha_i^* P$, $P_n^* = s_n^* bP$. We have

$$\begin{aligned} & e(V^*, P) \\ = \quad & e(R^*, E^*) e(U^*, T^*) e(P_n^*, F^*) \prod_{i=1}^{n} e(P_{i-1}^*, Q_i^*) \\ = \quad & e(R^*, \beta^* P) e(U^*, \pi^* P) e(s_n^* bP, \gamma^* aP) \prod_{i=1}^{n} e(P_{i-1}^*, \alpha_i^* P) \\ = \quad & e(\beta^* R^*, P) e(\pi^* U^*, P) e(s_n^* bP, \gamma^* aP) e(\sum_{i=1}^{n} \alpha_i^* P_{i-1}^*, P). \end{aligned}$$

$\mathcal{C}$ outputs

$$abP = (s_n^*\gamma^*)^{-1}(V^* - (\beta^* R^* + \pi^* U^* + \sum_{i=1}^{n} \alpha_i^* P_{i-1}^*))$$

as the solution of the CDH problem.

To complete the proof, we shall show that $\mathcal{C}$ solves the given instance of CDH problem with probability at least ϵ'. First, we analyze the three events needed for $\mathcal{C}$ to succeed:

- $\mathcal{E}4$: $\mathcal{C}$ does not abort as a result of any of $\mathcal{A}_{II}$'s Secret-Key Oracle, Public-Key-Replacement Oracle and Sign Oracle queries.
- $\mathcal{E}5$: σ^* is a valid signature on m^* under $(ID_0^*, ..., ID_n^*, P_0^*, ..., P_n^*)$.
- $\mathcal{E}6$: Both *Event 3* and *Event 4* happen.

$\mathcal{C}$ succeeds if all of these events happen. Hence, we have that

$$\epsilon' = \Pr[\mathcal{E}4 \wedge \mathcal{E}5 \wedge \mathcal{E}6] = \Pr[\mathcal{E}4]\Pr[\mathcal{E}5|\mathcal{E}4]\Pr[\mathcal{E}6|\mathcal{E}4 \wedge \mathcal{E}5].$$

Claim 4. The probability that $\mathcal{C}$ does not abort as a result of $\mathcal{A}_{II}$'s Secret-Key Oracle, Public-Key-Replacement Oracle and Sign Oracle queries is at least $(1-\delta)^{q_{SK}+q_{PKR}+q_S}$. Formally, we have $Pr[\mathcal{E}4] \geq (1-\delta)^{q_{SK}+q_{PKR}+q_S}$.

Proof. For a Secret-Key Oracle query, $\mathcal{C}$ will abort iff $coin_{PK} = 1$ happens. It is easy to see that the probability $\mathcal{C}$ does not abort for a Secret-Key Oracle query $\geq 1-\delta$. Since $\mathcal{A}_{II}$ can make at most q_{SK} times Secret-Key Oracle queries, the probability that $\mathcal{C}$ does not abort as a result of $\mathcal{A}_{II}$'s Secret-Key Oracle queries is at least $(1-\delta)^{q_{SK}}$.

For a Public-Key-Replacement Oracle query, $\mathcal{C}$ will abort iff $coin_{PK} = 1$ happens. Hence the probability that $\mathcal{C}$ does not abort for a Public-Key-Replacement Oracle query is $\geq 1-\delta$. Since $\mathcal{A}_{II}$ can make at most q_{PKR} times Public-Key-Replacement Oracle queries, the probability that $\mathcal{C}$ does not abort as a result of $\mathcal{A}_{II}$'s Public-Key-Replacement Oracle queries is at least $(1-\delta)^{q_{PKR}}$.

For a Sign Oracle query, $\mathcal{C}$ will abort iff $coin_{H_3} = 1$. It is easy to see that the probability that $\mathcal{C}$ does not abort for a Sign Oracle query is $\geq (1-\delta)$. Since $\mathcal{A}_{II}$ can make at most q_S times Sign Oracle queries, the probability that $\mathcal{C}$ does not abort as a result of $\mathcal{A}_{II}$'s Sign Oracle queries is at least $(1-\delta)^{q_S}$.

Hence, we have

$$\Pr[\mathcal{E}4] \geq (1-\delta)^{q_{SK}+q_{PKR}+q_S}.$$

□

Claim 5. $\Pr[\mathcal{E}5|\mathcal{E}4] \geq \epsilon$.
Proof. If Algorithm $\mathcal{C}$ does not abort as a result of $\mathcal{A}_{II}$'s Secret-Key Oracle, Public-Key-Replacement Oracle and Sign Oracle queries then Algorithm $\mathcal{A}_{II}$'s view is identical to its view in the real attack. Hence, $\Pr[\mathcal{E}5|\mathcal{E}4] \geq \epsilon$. □

Claim 6. The probability that $\mathcal{C}$ does not abort after $\mathcal{A}_{II}$ outputting a valid forgery is at least δ^2. Formally, we have $\Pr[\mathcal{E}6|\mathcal{E}4 \wedge \mathcal{E}5] \geq \delta^2$.
Proof. If events $\mathcal{E}4$ and $\mathcal{E}5$ have occurred, $\mathcal{C}$ will abort unless *Event 3* or *Event 4* happen. If *Event 3* happens, that means $\mathcal{C}$ generates a valid message-signature pair (m^*, σ^*) under $(ID_0^*, ..., ID_n^*, P_0^*, ..., P_n^*)$ without querying the corresponding hash functions. This probability is negligible. On the other hand, $\Pr[\mathit{Event}\ 4] = \Pr[coin_{H_3}^* = coin_{PK}^* = 1] \geq \delta^2$. Therefore, $\Pr[\mathcal{E}6|\mathcal{E}4 \wedge \mathcal{E}5] \geq \delta^2$. □

Altogether, we have

$$\epsilon' = \Pr[\mathcal{E}4 \wedge \mathcal{E}5 \wedge \mathcal{E}6] \geq (1-\delta)^{q_{SK}+q_{PKR}+q_S}\delta^2\epsilon.$$

When $\delta = \frac{2}{q_{SK}+q_{PKR}+q_S+2}$, we have

$$\begin{aligned}\epsilon' \geq\ & (1 - \frac{2}{q_{SK}+q_{PKR}+q_S+2})^{q_{SK}+q_{PKR}+q_S} \cdot \\ & (\frac{2}{q_{SK}+q_{PKR}+q_S+2})^2\epsilon,\end{aligned}$$

This completes the proof. □

Theorem 3. *Our scheme is a secure HCLS scheme.*

Proof. An HCLS scheme is secure if it is secure against Type I and II adversaries. Hence, this theorem follows directly from Theorem 1 and 2.

5. Discussion

In this section, we discuss generic constructions of HCLSs and a short HCLS scheme.

5.1. Generic Construction

We distinguish our work from potential generic constructions of HCLS schemes. In [25], Yum and Lee presented a generic way to construct a certificateless signature scheme. Subsequently, Hu et al. [16] pointed out that the Yum-Lee construction is flawed and proposed a new one. It seems at first glance that Hu et al.'s methods can also be used to obtain a generic construction of HCLSs by combination of hierarchical identity-based signature and normal public key signature. Roughly speaking, in this generic construction, a signer firstly generates a signature σ on the message m using a normal public key signature scheme, then the signer generates a signature σ' on m and σ using an hierarchical identity-based signature scheme. Unfortunately, for a construction of this kind, the scheme has the weakness of how to detect dishonest behavior and how to identify a malicious KGC. In other words, if a malicious (root or lower-level) KGC forges a user's signature, any KGC may deny its dishonest behavior. On the contrary, our scheme does not suffer from this weakness.

5.2. Short HCLS Scheme

It is interesting to achieve short HCLS schemes. HCLC was first studied in [1], in which a hierarchical certificateless encryption scheme was instantiated. It seems that this hierarchical certificateless encryption scheme implies an HCLS scheme with short signature output. The following HCLS scheme is derived from the hierarchical certificateless encryption scheme in [1].

- Root-Setup: On input 1^λ, this algorithm runs as follows:
  1. Choose $\mathbb{G}_1, \mathbb{G}_2, e$ as defined in Section 3.1.
  2. Choose a random generator $P \in \mathbb{G}_1$.
  3. Select a *root* private key $s_0 \in \mathbb{Z}_q^*$ at random and set $P_0 = s_0 P$ as the public key of the root KGC $\mathcal{K}_0$.
  4. Choose cryptographic hash functions $H_1, H_2 : \{0,1\}^* \longrightarrow \mathbb{G}_1$.

  The system parameters are $params = (\mathbb{G}_1, \mathbb{G}_2, e, P, P_0, H_1, H_2, ID_0, \mathcal{M})$, where ID_0 is the identity of $\mathcal{K}_0$ and $\mathcal{M} = \{0,1\}^*$ is the message space.

- Lower-Level-Setup: The same as that in Section 3.2.

- Set-Secret-Value: The same as that in Section 3.2.

- Set-Public-Key: The same as that in Section 3.2.

- Partial-Private-Key-Extract: On input $\mathcal{U}_n$'s identifying information $(ID_0, ..., ID_n)$ and the corresponding public key list $(P_0, ..., P_n)$, $\mathcal{K}_{n-1}$ computes the partial private key for $\mathcal{U}_n$ as follows:
  1. Compute $Q_n = H_1(ID_0, ..., ID_n, P_0, ..., P_n)$.
  2. Compute $D_n = D_{n-1} + s_{n-1}Q_n = \sum_{i=1}^{t} s_{i-1}Q_i$.

3. Output the partial private key D_n.

- Set-Private-Key: On input $\mathcal{U}_n$'s secret value s_n, partial private key D_n, identifying information $(ID_0, ..., ID_n)$ and the public key list $(P_0, ..., P_n)$, $\mathcal{U}_n$ sets $S_n = (s_n, D_n)$ as her private key.

- Sign: Let the signer $\mathcal{U}_n$ have identifying information $(ID_0, ..., ID_n)$, public keys $(P_0, ..., P_n)$ and private key $S_n = (s_n, D_n)$. To sign a message $m \in \mathcal{M}$, $\mathcal{U}_n$ uses the private key S_n and performs the following steps of which the first four can be pre-computed:

  1. Compute $Q_i = H_1(ID_0, ..., ID_i, P_0, ..., P_i)$ for $1 \leq i \leq n$.
  2. Compute $F = H_2(ID_0, ..., ID_n, P_0, ..., P_n, m)$.
  3. Compute $\sigma = D_n + s_n F$.
  4. Output σ as the signature on m.

- Verify: To verify a signature σ on message m under $(ID_0, ..., ID_n, P_0, ..., P_n)$, the verifier performs the following steps:

  1. Compute $Q_i = H_1(ID_0, ..., ID_i, P_0, ..., P_i)$ for $1 \leq i \leq n$.
  2. Compute $F = H_2(ID_0, ..., ID_n, P_0, ..., P_n, m)$.
  3. Check whether

$$e(\sigma, P) \overset{?}{=} e(P_n, F) \prod_{i=1}^{n} e(P_{i-1}, Q_i)$$

  holds with equality. Output *true* if the equality holds; otherwise, output *false*.

The above construction is more efficient than the scheme in Section 3.2 and has short signature size, i.e., a signature only consists of one group element. However, we remark that the above scheme can only be proven secure in a weaker security model in which a Type I adversary is a normal Type I adversary [17]. In fact, when the hierarchical level is 1 (i.e., the scheme degenerates to the normal CLS), the scheme is exactly the short signature scheme in [17] that is secure against normal Type I and super Type II adversaries. We note that a normal Type I adversary is a weaker adversary than those in the real world. If a CLS scheme is only secure against normal Type I adversary and a user misuses her private key (e.g., signs a message with a replaced public key), then a Type I adversary can recover the full private key of the user. Therefore, it remains an open problem to construct short HCLS schemes secure against super Type I and II adversaries.

6. Conclusion

We have introduced hierarchical certificateless signature schemes and their related security definitions. In our security definitions, we treat Type I and Type II adversaries as super adversaries. We have proposed a concrete fully scalable HCLS scheme where the signatures consist of three group elements. The proposed scheme is provably secure against both types of super adversaries under the CDH assumption in the random oracle model. With our scheme, even if an attacker colludes with other users in the system, he cannot forge a valid HCLS on behalf of his target user; also, if a KGC forges an HCLS, then this dishonest behavior can be detected. We have also distinguished our scheme from a possible generic construction and a construction with short signatures, and shown that our scheme has a higher security level than the other two constructions.

Acknowledgments and disclaimer

This work was supported in part by the NSF of China under grants 61202465, 61021004, 11061130539, 61103222, 61173154 and 61003214; the Shanghai NSF under grant no. 12ZR1443500; the Shanghai Chen Guang Program (12CG24); EU FP7 under projects "DwB" and "Inter-Trust"; the Spanish Government under projects TIN2011-27076-C03-01 and CONSOLIDER INGENIO 2010 "ARES" CSD2007-0004; the Government of Catalonia under grant SGR2009-1135; the Fundamental Research Funds for the Central Universities of China; the Open Project of Shanghai Key Laboratory of Trustworthy Computing (no. 07dz22304201101). J. Domingo-Ferrer was supported in part as an ICREA-Acadèmia researcher by the Government of Catalonia.

References

[1] S. Al-Riyami and K. Paterson, Certificateless public key cryptography. ASIACRYPT 2003, LNCS 2894, pp. 452-473, 2003.

[2] M. Au, J. Chen, J. Liu, Y. Mu, D. Wong and G. Yang, Malicious KGC attacks in certificateless cryptography. ACM ASIACCS, pp. 302-311, 2007.

[3] M. Bellare and P. Rogaway, Random oracles are practical: A paradigm for designing efficient protocols. ACM CCS 1993, pp. 62-73, 1993.

[4] A. Boldyreva, V. Goyal and V. Kumar, Identity-based encryption with efficient revocation. ACM CCS 2008, pp. 417-426, 2008.

[5] D. Boneh, X. Boyen, and E. Goh, Hierarchical identity based encryption with constant size ciphertext. EUROCRYPT 2005, LNCS 3494, pp. 440-456, 2005.

[6] D. Boneh and M. Franklin, Identity-based encryption from the Weil pairing. SIAM Journal on Computing, 32(3): 586-615, 2003.

[7] X. Boyen and B. Waters, Anonymous hierarchical identity-based encryption (without random oracles). CRYPTO 2006, LNCS 4117, pp. 290-307, 2006.

[8] X. Boyen and B. Waters, Compact group signatures without random oracles. EUROCRYPT 2006, LNCS 4004, pp. 427-444, 2006.

[9] R. Canetti, S. Halevi, and J. Katz, A forward-secure public-key encryption scheme. EUROCRYPT 2003, LNCS 2656, pp. 255-271, 2003.

[10] A. Dent, B. Libert, and K. Paterson, Certificateless encryption schemes strongly secure in the standard model. PKC 2008, LNCS 4939, pp. 344-359, 2008.

[11] M. Girault, Self-certified public keys. EUROCRYPT'91, LNCS 547, pp. 490-497, 1992.

[12] C. Gentry and A. Silverberg, Hierarchical ID-based cryptography. ASIACRYPT 2002, LNCS 2501, pp. 548-566, 2002.

[13] R. Granger and N.P. Smart, On computing products of pairings. Cryptology ePrint Archive: Report 2006/172. http://eprint.iacr.org/2006/172.

[14] Y. Hanaoka, G. Hanaoka, J. Shikata, H. Imai, Identity-based hierarchical strongly key-insulated encryption and its application. Cryptology ePrint Archive: Report 2004/338. http://eprint.iacr.org/2004/338.

[15] J. Horwitz and B. Lynn, Towards hierarchical identity-based encryption. EUROCRYPT 2002, LNCS 2332, pp. 466-481, 2002.

[16] B. Hu, D. Wong, Z. Zhang and X. Deng, Key replacement attack against a generic construction of certificateless signature. ACISP 2006, LNCS 4058, pp. 235-346, 2006.

[17] X. Huang, Y. Mu, W. Susilo, D. Wong, and W. Wu, Certificateless signature revisited. ACISP 2007, LNCS 4586, pp. 308-322, 2007.

[18] X. Huang, W. Susilo, Y. Mu and F. Zhang, On the security of a certificateless signature scheme. CANS 2005, LNCS 3810, pp. 13-25, 2005.

[19] J. Li, Z. Wang, Y. Zhang, Provably secure certificate-based signature scheme without pairings, Information Sciences, 233: 313-320, 2013.

[20] H. Lim and K. Paterson, Multi-key hierarchical identity-based signatures. Cryptography and Coding 2007, LNCS 4887, pp. 384-402, 2007.

[21] S. Miao, F. Zhang, S. Li, Y. Mu, On security of a certificateless signcryption scheme, Information Sciences, 232: 475-481, 2013.

[22] V. Miller, The Weil pairing, and its efficient calculation. Journal of Cryptology, 17(4): 235-261, 2004.

[23] Y. Ren and D. Gu, Secure hierarchical identity based encryption scheme in the standard model. INDOCRYPT 2008, LNCS 5365, pp. 104-115, 2008.

[24] A. Shamir, Identity based cryptosystems and signature schemes, Crypto'84, LNCS 196, pp. 47-53, 1984.

[25] D. Yum and P. Lee, Generic construction of certificateless signature, ACISP 2004, LNCS 3108, pp. 200-211, 2004.

[26] J. Zhang, J. Mao, An efficient RSA-based certificateless signature scheme. Journal of Systems and Software, 85(3): 638-642, 2012.

[27] Z. Zhang, D. Wong, J. Xu, D. Feng, Certificateless public-key signature: security model and efficient construction. ACNS 2006, LNCS 3989, pp. 293-308, 2006.

[28] L. Zhang, Q. Wu, J. Domingo-Ferrer, B. Qin, Hierarchical certificateless signatures. TRUSTCOM 2010, pp. 572-577, 2010.

[29] L. Zhang, F. Zhang, B. Qin, S. Liu, Provably-secure electronic cash based on certificateless partially-blind signatures. Electronic Commerce Research and Applications, 10(5): 545-552, 2011.

[30] L. Zhang, F. Zhang, Q. Wu, Delegation of signing rights using certificateless proxy signatures. Information Sciences, 184(1): 298-309, 2012.

[31] L. Zhang, F. Zhang, Q. Wu, J. Domingo-Ferrer, Simulatable certificateless two-party authenticated key agreement protocol. Information Sciences, 180(6): 1020-1030, 2010.

[32] L. Zhang, B. Qin, Q. Wu, F. Zhang, Efficient many-to-one authentication with certificateless aggregate signatures, Computer Networks, 54(14): 2482-2491, 2010.