Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

Certificateless aggregate signature scheme secure against fully chosen-key attacks

Ge Wu[a,c,d], Futai Zhang[b,*], Limin Shen[b], Fuchun Guo[c], Willy Susilo[c]

[a] School of Cyber Science and Engineering, Southeast University, Nanjing, China
[b] School of Computer Science and Technology, Nanjing Normal University, Nanjing, China
[c] Institute of Cybersecurity and Cryptology, School of Computing and Information Technology, University of Wollongong, Wollongong, Australia
[d] Purple Mountain Laboratories, Nanjing, China

A R T I C L E I N F O

Article history:
Received 28 January 2019
Revised 16 October 2019
Accepted 24 November 2019
Available online 25 November 2019

Keywords:
Certificateless aggregate signature
Security model
Fully chosen-key attacks
Malicious KGC attacks

A B S T R A C T

Certificateless aggregate signature (CLAS) schemes enjoy the benefits of both certificateless cryptography and aggregate signature features. Specifically, it not only simplifies the certificate management without introducing the key escrow problem but also transforms many signatures into one aggregate signature to save communication and computation cost.

CLAS is a powerful cryptographic tool, yet its security should be thoroughly analyzed before being implemented. In this paper, we give a new insight into the security of CLAS schemes. We introduce a potential and realistic attack called *fully chosen-key attacks* that has not been considered in the traditional security models and define the security model against *fully chosen-key attacks*. In contrast to the traditional models, the adversary is allowed to hold all the signers' private keys and its goal is not to forge an aggregate signature but to output invalid single signatures that can be aggregated into a valid aggregate signature. We find there is no CLAS scheme secure in traditional security models that is secure against fully chosen-key attacks and then demonstrate how to reinforce the security of an existing scheme to withstand such an attack.

1. Introduction

Certificateless cryptography [1] was introduced by Riyami and Paterson to deal with the complicated certificate management problem in the traditional Public-Key Infrastructure (PKI) setting without bringing the key-escrow problem. Aggregate signature [3] can be employed to transform many signatures into one aggregate signature. The motivation of the aggregate signature technique is to reduce the communication and computation costs, where only the aggregate signature is transmitted and verified instead of all the single signatures involved in the aggregation. In situations where a large number of signatures are required to be verified, such as vehicular networks [6,11], audit log of a computer system [13], auditing scheme with group users [7], and Secure Border Gateway Protocol (SBGP) [24], aggregate signature is a powerful tool to achieve efficiency improvement.

[*] Corresponding author.
E-mail address: zhangfutai@njnu.edu.cn (F. Zhang).

https://doi.org/10.1016/j.ins.2019.11.037
0020-0255/© 2019 Elsevier Inc. All rights reserved.

Certificateless aggregate signature (CLAS) schemes enjoy the advantages of both certificateless cryptography and aggregate signature features. It has been found that various applications, not limited to the aforementioned ones, since it was first introduced in [8]. For example, Cloud Storage is gaining its popularity due to low hardware cost, large storage volume, and easy access control. On the other hand, the stability of the storage service is the main security issue that has always been prioritized by users. Cloud auditing is a cryptographic solution for this purpose and in particular, CLAS is a suitable technique to improve the efficiency of the cloud auditing process.

The main entities in a Cloud Storage platform implementing a certificateless cryptographic scheme include the Key Generation Center (KGC), cloud server, Third-Party Auditor (TPA), and the users. During an auditing process, the cloud server computes the response to the challenge from the TPA. Since there are many users in the system, the TPA can send multiple challenges on behalf of multiple users to the cloud server. In a straightforward auditing process, the cloud server computes the corresponding responses and returns them to the TPA. Then, the TPA verifies all the responses one by one. If the cloud server aggregates the responses, both the communication and computation costs of transmitting and verifying the responses can be greatly saved. CLAS technique is suitable for many auditing schemes.

1.1. Related work

The notion of certificateless aggregate signature (CLAS) schemes was put forth in [8] and many practical constructions [6,9,10,14–16,18,23,25] are available in the literature to date. In certificateless cryptography, two types of adversaries, namely type I adversary and type II adversary are considered. Furthermore, Huang et al. [12] classified each type of adversary into three kinds; namely normal adversary, strong adversary, and super adversary. Au et al. [2] considered a more powerful KGC (type II) adversary called *Malicious-But-Passive KGC* (malicious KGC for short). A summary of these security notions can be found in [21].

Recently, Xiong et al. put forward a CLAS scheme [24] with constant pairing computations. Later, cryptanalyses of this construction have been presented [5,10,25]. Cheng et al. [5] showed that the single signature of the scheme in [24] does not achieve unforgeability against *Honest-But-Curious KGC* adversary and proposed an improved scheme to withstand malicious KGC adversary. The security model of CLAS schemes was formalized by Zhang et al. in [28] and has been widely considered for the security model of subsequent scheme constructions and security analyses [4,5,10,19,20,22,25–27]. The model extends the definition of the seminal work of aggregate signature by Boneh et al. [3] to the certificateless cryptography setting. Although the security model guarantees that the adversary cannot forge an aggregate signature on behalf of all the signers, if it does not hold all the signers' private keys, we show in this paper that this does not capture all potential threats in practical applications.

1.2. Revisiting the security of aggregate signature

The security considered for a signature scheme is unforgeability. Specifically, it guarantees that an adversary without accessing the user's private key cannot forge a signature on behalf of the user. As for the aggregate signature setting, n signatures signed by n users are aggregated into one aggregate signature. A straightforward extension of the security to the aggregate signature setting is the unforgeability of the aggregate signature. To be more precise, an adversary without accessing all the n users' private keys cannot forge an aggregate signature on behalf of the users. As a result, the adversary is allowed to adaptively hold at most $n-1$ users' private keys and its goal is to forge a valid aggregate signature on behalf of the n users. This is the security model considered in the seminal work of aggregate signature [3], which is referred to as the *"chosen-key" security model*. The *existential unforgeability against chosen-key attacks and chosen-message attacks* has become the standard security notion of aggregate signature schemes. However, as we shall show in this work, this security does not capture all potential attacks in practical applications.

Fully chosen-key attacks. The motivation of the aggregate signature technique is to improve the efficiency of transmitting and verifying a large number of signatures. In an aggregate signature scheme, only the aggregate signature instead of all single signatures involved in the aggregation is transmitted and verified. The verifier wants to check the validities of all single signatures by verifying the aggregate signature. To convince the verifier, the aggregate signature scheme should provide a guarantee that the validity of the aggregate signature is equivalent to the validities of all the single signatures. In this paper, we refer to it as the *equivalent validity*.

A secure aggregate signature scheme should achieve *equivalent validity* not only *unforgeability*. In traditional security models against *chosen-key attacks*, the adversary holds at most $n-1$ private keys amongst n users. However, what if the adversary holds all the users' private keys and its goal is not to forge an aggregate signature but to break the equivalent validity. For example, two dishonest users work together (of course they hold both their private keys) to cheat the verifier. They output two invalid single signatures that are aggregated into a valid aggregate signature. Note that this is a practical attack which breaks the equivalent validity of the aggregate signature scheme. On one hand, the verifier believes that the two single signatures are valid after verifying the aggregate signature. On the other hand, the single signatures are invalid and the two dishonest users can deny that they have signed the signatures. In other words, the equivalent validity should be guaranteed no matter whether the adversary holds all the users' private keys or not. In this paper, we allow the adversary to hold all the users' private keys and call it *fully chosen-key attacks* in contrast to the traditional security models.

To sum up, a secure aggregate signature should satisfy security requirements in two aspects. Namely, the *existential unforgeability* of single signatures against *chosen-message attacks* and *equivalent validity* of an aggregate signature against *fully chosen-key attacks*. In particular, for a certificateless aggregate signature (CLAS) scheme, the single signatures should be existentially unforgeable against both type I and type II adversaries. Since the fully chosen-key attacks allow the adversary to hold all the users' private keys, there is no need to distinguish the two types of adversaries in the security model of *equivalent validity against fully chosen-key attacks*.

1.3. Our contributions

In this paper, we take an insight into the security of certificateless aggregate signature (CLAS) schemes. The contributions of this work are three-fold and summarized as follows.

- We consider the *equivalent security against fully chosen-key attacks* which is not captured in the traditional security models of CLAS schemes and propose the formal definition and security model.
- A scheme secure in the traditional security model is not necessarily secure against *fully chosen-key attacks*. We demonstrate how to reinforce the security to withstand fully chosen-key attacks taking the scheme in [5] as an example. The transformation is general, where a collision-resistant hash function is applied.
- In addition, we give re-formalized security proofs of the single signatures in the proposed scheme, which include the *existential unforgeability against chosen-message attacks* by a strong type I adversary and a super type II (Malicious-But-Passive KGC) adversary.

1.4. Organization

The rest of this paper is organized as follows.

We begin with presenting the definition and security models of certificateless aggregate signature (CLAS) schemes in Section 2. A review of the scheme in [5] is given in Section 3. In Section 4, we propose an improved scheme, secure in the security models defined in this paper. The efficiency and experimental analyses will be provided in Section 5. Finally, we conclude this paper in Section 6.

2. Definition and security models

In this section, we present the definition and define the security models of certificateless aggregate signature (CLAS) schemes.

As analyzed in Section 1.2, a CLAS scheme should satisfy two aspects of security requirements, namely the *existential unforgeability against chosen-message attacks* of single signatures and the *equivalent validity against fully chosen-key attacks* of aggregate signatures. As for the security model of existential unforgeability, type I and type II adversaries are considered. Following [12], we further divide each type of adversary into three kinds; namely normal, strong, and super. In this paper, a type II adversary refers to a stronger definition of Malicious-But-Passive KGC (malicious KGC for short) by Au et al. [2]. In particular, we consider the security models of *existential unforgeability against chosen-message attacks* by a strong type I adversary and a super type II (malicious KGC) adversary. The detailed definitions of the strong/super adversaries and malicious KGC adversary can be found in [2,12], respectively.

The security model of *equivalent validity against fully chosen-key attacks* defined in this paper gives the adversary more powerful attacking capability, where it is allowed to hold all the users' private keys. Therefore, there is no need to distinguish the type I and type II adversaries.

2.1. Definition of CLAS schemes

In many applications that employ the aggregate signature technique, there is an intended verifier that is responsible for verifying the aggregate signatures, such as the Third Party Auditor (TPA) in a cloud auditing system. Normally, this verifier has been deployed in advance when the system is set up. To reflect this fact, we define the CLAS scheme that additionally takes as input the verifier's public key in generating the aggregate signature and the verifier's secret key in verifying the aggregate signature. Therefore, any user can generate an aggregate signature for the intended verifier and only the verifier can verify the validity of the aggregate signature.

A CLAS scheme comprises seven algorithms; namely (i) the master key generation algorithm, (ii) partial private key generation algorithm, (iii) user key generation algorithm, (iv) signing algorithm, (v) verification algorithm, (vi) aggregation algorithm, and (vii) aggregation verification algorithm. These algorithms are denoted by Setup, $\mathsf{PartialKeyGen}$, $\mathsf{UserKeyGen}$, Sign, Verify, $\mathsf{Aggregate}$, and AggVer, respectively. The detailed description is as follows.

- $\mathsf{Setup}(1^\lambda)$: The master key generation algorithm takes as input the security parameter λ and outputs the master public and secret key pair (mpk, msk). After the master public key mpk has been generated, the intended verifier in the system generates a public and secret verification key pair $(pk_{\mathsf{ver}}, sk_{\mathsf{ver}})$ and publishes pk_{ver}.

- $\mathsf{PartialKeyGen}(mpk, msk, \mathsf{ID})$: The partial private key generation algorithm takes as input the master public key *mpk*, master secret key *msk*, and a user's identity ID. It outputs a partial private key d_{ID} of the user.
- $\mathsf{UserKeyGen}(mpk, \mathsf{ID})$: The user key generation algorithm takes as input the master public key *mpk* and a user's identity ID. It outputs a pair of public key and secret value $(pk_{\mathsf{ID}}, \nu_{\mathsf{ID}})$.
- $\mathsf{Sign}(mpk, sk_{\mathsf{ID}}, m)$: The signing algorithm takes as input the master public key *mpk*, private key $sk_{\mathsf{ID}} = (d_{\mathsf{ID}}, \nu_{\mathsf{ID}})$ of the user with identity ID, and a message m. It outputs a signature σ on message m signed by the user with identity ID.
- $\mathsf{Verify}(mpk, \mathsf{ID}, pk_{\mathsf{ID}}, m, \sigma)$: The verification algorithm takes as input the master public key *mpk*, user's identity ID, public key pk_{ID}, a message m, and a signature σ. It outputs the verification result of σ, i.e. "Accept" or "Reject".
- $\mathsf{Aggregate}(mpk, \{(\mathsf{ID}_i, pk_{\mathsf{ID}_i}, m_i, \sigma_i) \mid i = 1, \ldots, n\}, pk_{\mathsf{ver}})$: The aggregation algorithm takes as input the master public key *mpk*, a set of four-tuples $\{(\mathsf{ID}_i, pk_{\mathsf{ID}_i}, m_i, \sigma_i) \mid i = 1, \ldots, n\}$, consisting of identities, public keys, messages, signatures, and the public verification key pk_{ver}. It outputs the aggregate signature Σ of the single signatures.
- $\mathsf{AggVer}(mpk, \{(\mathsf{ID}_i, pk_{\mathsf{ID}_i}, m_i) \mid i = 1, \ldots, n\}, \Sigma, sk_{\mathsf{ver}})$: The aggregation verification algorithm takes as input the master public key *mpk*, a set of trituples $\{(\mathsf{ID}_i, pk_{\mathsf{ID}_i}, m_i) \mid i = 1, \ldots, n\}$, consisting of identities, public keys, messages, an aggregate signature Σ, and the secret verification key sk_{ver}. It outputs the verification result of Σ, i.e. "Accept" or "Reject".

2.2. Security model of equivalent validity

The equivalent validity of the aggregate signature requires that the aggregate signature is valid if and only if all the single signatures involved in the aggregation are valid. A secure CLAS scheme should provide such a guarantee no matter whether the adversary holds all the users' private keys or not. The security model of *equivalent validity against fully chosen-key attacks* allows the adversary to hold all the users' private keys. Since verifying the aggregate signature needs the secret key of the intended verifier, we provide the adversary an additional oracle to check the validity of the aggregate signatures. The adversary's goal is to output invalid single signatures that can be aggregated into a valid aggregate signature. The security model is defined as the following game between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$.

Setup. $\mathcal{C}$ runs the master key generation algorithm Setup to generate the master public and secret key pair (*mpk, msk*). Then, it generates a public and secret verification key pair $(pk_{\mathsf{ver}}, sk_{\mathsf{ver}})$ of the intended verifier. Next, it returns the master public key *mpk* and public verification key pk_{ver} to $\mathcal{A}$.

Private key query. $\mathcal{A}$ can adaptively query the private key of the user with identity ID. $\mathcal{C}$ returns the private key $sk_{\mathsf{ID}} = (d_{\mathsf{ID}}, \nu_{\mathsf{ID}})$ which includes the partial private key d_{ID} and secret value ν_{ID} to $\mathcal{A}$. If d_{ID} or ν_{ID} has not been generated, $\mathcal{C}$ runs the partial private key generation algorithm $\mathsf{PartialKeyGen}$ or user key generation algorithm $\mathsf{UserKeyGen}$.

Aggregation verification query. $\mathcal{A}$ can adaptively make aggregation verification queries by submitting a set of trituples $\{(\mathsf{ID}_i, pk_{\mathsf{ID}_i}, m_i) \mid i = 1, \ldots, n\}$ consisting of identities, public keys, messages, and an aggregate signature Σ. $\mathcal{C}$ runs the aggregation verification algorithm AggVer with the secret verification key sk_{ver} as input and returns the verification result.

Forgery. At last, $\mathcal{A}$ outputs a set of four-tuples $\{(\mathsf{ID}_i, pk_{\mathsf{ID}_i}, m_i, \sigma_i) \mid i = 1, \ldots, n\}$ consisting of identities, public keys, messages, signatures, and a value Σ supposed to be the aggregate signature on the set of four-tuples.

We say $\mathcal{A}$ wins the game if the following conditions are satisfied.

1. The aggregate signature Σ is generated from all the single signatures,
$$\mathsf{Aggregate}(mpk, \{(\mathsf{ID}_i, pk_{\mathsf{ID}_i}, m_i, \sigma_i) \mid i = 1, \ldots, n\}, pk_{\mathsf{ver}}) = \Sigma.$$
2. The aggregate signature Σ is valid, i.e.
$$\mathsf{AggVer}(mpk, \{(\mathsf{ID}_i, pk_{\mathsf{ID}_i}, m_i) \mid i = 1, \ldots, n\}, \Sigma, sk_{\mathsf{ver}}) = \text{“Accept”}.$$
3. There exists at least one invalid single signature, i.e.
$$\exists\, i^* \in \{1, \ldots, n\},\ \ s.t.\ \mathsf{Verify}(mpk, \mathsf{ID}_{i^*}, pk_{\mathsf{ID}_{i^*}}, m_{i^*}, \sigma_{i^*}) = \text{“Reject”}.$$

3. An example of fully chosen-key attacks

In this section, we use the certificateless aggregate signature (CLAS) scheme in [5] as an example to illustrate the fully chosen-key attacks. In addition, we make a slight modification of the scheme to improve it to withstand the malicious KGC attacks. This modification is essential for the security proof of the existential unforgeability of single signatures.

3.1. Mathematical preliminaries

A (symmetric) bilinear pairing is defined over two cyclic groups of the same prime order. The pairing groups denoted by $\mathbb{PG} = (\mathbb{G}, \mathbb{G}_T, q, e)$ include groups $\mathbb{G}, \mathbb{G}_T$ of order q, where e denotes a bilinear map $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$.

Suppose group $\mathbb{G}$ is additive and P is a generator, the computational Diffie-Hellman (CDH) assumption in group $\mathbb{G}$ is as follows. Given elements (P, aP, bP), for random $a, b \in \mathbb{Z}_q$, it is intractable to compute abP.

A collision-resistant hash function $H : \mathcal{X} \to \mathcal{Y}$ maps a message in the input space $\mathcal{X}$ into a fixed-size message digest in the output space $\mathcal{Y}$ that satisfies the collision-resistant property. Namely, given a hash function H, it is intractable to find a pair of messages (x, x'), such that $x \neq x'$ and $H(x) = H(x')$.

3.2. Scheme description

The scheme construction in [5] is reviewed as follows.

Note that there does not exist a verifier who is responsible for verifying the aggregate signatures in this construction. Therefore, the inputs of the aggregation algorithm and aggregation verification algorithm do not include the public verification key and secret verification key, respectively.

- $\mathsf{Setup}(1^\lambda)$: The KGC runs the master key generation algorithm with the security parameter λ as input. It chooses pairing groups $\mathbb{PG} = (\mathbb{G}, \mathbb{G}_T, q, e)$ and a generator $P \in \mathbb{G}$. Then, it selects three hash functions $H_1, H_2 : \{0,1\}^* \to \mathbb{G}$ and $H_3 : \{0,1\}^* \to \mathbb{Z}_q$. Next, it randomly picks a bit string Q of length l, an element $s \in \mathbb{Z}_q$, and computes $P_{pub} = sP$. It outputs the master public key $mpk = (\mathbb{PG}, l, P, Q, P_{pub}, H_1, H_2, H_3)$ and keeps the master secret key $msk = s$ secretly.
- $\mathsf{PartialKeyGen}(mpk, msk, \mathsf{ID})$: The KGC runs the partial private key generation algorithm with the master public key mpk, master secret key $msk = s$, and a user's identity ID as input. It computes $H_2(\mathsf{ID}) \in \mathbb{G}$ and $d_{\mathsf{ID}} = s \cdot H_2(\mathsf{ID})$. Then, it outputs d_{ID} as the partial private key of user ID.
- $\mathsf{UserKeyGen}(mpk, \mathsf{ID})$: The user with identity ID runs the user key generation algorithm with the master public key mpk and identity ID as input. It selects a random secret value $x \in \mathbb{Z}_q$ and computes $pk_{\mathsf{ID}} = xP$. It sets the secret value $\nu_{\mathsf{ID}} = x$ and outputs $(pk_{\mathsf{ID}}, \nu_{\mathsf{ID}})$ as a pair of public key and secret value.
- $\mathsf{Sign}(mpk, sk_{\mathsf{ID}}, m)$: The user with identity ID runs the signing algorithm with the master public key mpk, private key $sk_{\mathsf{ID}} = (d_{\mathsf{ID}}, \nu_{\mathsf{ID}})$, and a message m as input. It performs as follows:
  1. Choose a random element $r \in \mathbb{Z}_q$ and compute $U = rP$.
  2. Compute $h = H_3(m, \mathsf{ID}, pk_{\mathsf{ID}}, U) \in \mathbb{Z}_q$ and
  $$V = d_{\mathsf{ID}} + h \cdot r \cdot P_{pub} + h \cdot \nu_{\mathsf{ID}} \cdot H_1(Q) + r \cdot H_1(Q).$$
  3. Set $\sigma = (U, V)$.

  It outputs σ as a signature on message m signed by user ID.
- $\mathsf{Verify}(mpk, \mathsf{ID}, pk_{\mathsf{ID}}, m, \sigma)$: Any user can run the verification algorithm with the master public key mpk, identity ID, public key pk_{ID}, message m, and a signature $\sigma = (U, V)$ as input. It first computes $h = H_3(m, \mathsf{ID}, pk_{\mathsf{ID}}, U)$ and then checks whether the following equation holds.
  $$e(V, P) = e\Big(H_2(\mathsf{ID}) + h \cdot U, P_{pub}\Big) \cdot e\Big(h \cdot pk_{\mathsf{ID}} + U, H_1(Q)\Big)$$
  If yes, it outputs "Accept". Otherwise, it outputs "Reject".
- $\mathsf{Aggregate}(mpk, \{(\mathsf{ID}_i, pk_{\mathsf{ID}_i}, m_i, \sigma_i) \mid i = 1, \ldots, n\})$: Any user can run the aggregation algorithm with the master public key mpk and a set of four-tuples $(\mathsf{ID}_i, pk_{\mathsf{ID}_i}, m_i, \sigma_i)$ consisting of identities, public keys, message, and signatures as input, where $\sigma_i = (U_i, V_i)$. It computes $V = \sum_{i=1}^{n} V_i$ and outputs $\Sigma = (U_1, \ldots, U_n, V)$ as the aggregate signature on the messages signed by the identities and public keys in the set of four-tuples.
- $\mathsf{AggVer}(mpk, \{(\mathsf{ID}_i, pk_{\mathsf{ID}_i}, m_i) \mid i = 1, \ldots, n\}, \Sigma)$: Any user can run the aggregation verification algorithm with the master public key mpk, a set of trituples $(\mathsf{ID}_i, pk_{\mathsf{ID}_i}, m_i)$ consisting of identities, public keys, messages, and an aggregate signature $\Sigma = (U_1, \ldots, U_n, V)$ as input. It first computes $h_i = H_3(m_i, \mathsf{ID}_i, pk_{\mathsf{ID}_i}, U_i)$, for $i = 1, \ldots, n$ and then checks whether the following equation holds
  $$e(V, P) = e\left(\sum_{i=1}^{n}\Big(H_2(\mathsf{ID}_i) + h_i \cdot U_i\Big), P_{pub}\right) \cdot e\left(\sum_{i=1}^{n}(h_i \cdot pk_{\mathsf{ID}_i} + U_i), H_1(Q)\right)$$
  If yes, it outputs "Accept". Otherwise, it outputs "Reject".

Remark 1. To overcome the drawback of the scheme in [24], the scheme in [5] applies an additional hash function $H_1 : \{0,1\}^* \to \mathbb{G}$. Instead of randomly selecting generator $Q \in \mathbb{G}$, it first picks a string Q of length l and computes the generator as $H_1(Q) \in \mathbb{G}$. However, this modification is not enough to withstand the malicious KGC attacks. The reason is that there is a prerequisite, namely the discrete logarithm of $H_1(Q)$ to the base P should be unknown to the adversary to prevent the malicious KGC attacks.

Unfortunately, as a malicious adversary (who is able to generate the master public key), it can first select the string Q, computes $H_1(Q)$, and sets $P = \alpha H_1(Q)$, for some $\alpha \in \mathbb{Z}_q$, such that it actually knows the discrete logarithm. This is where the security proof (lemma 2) in [5] does not hold. In detail, the simulator needs to embed the CDH problem instance into the master public key mpk while mpk is chosen by the adversary.

To fix this issue, we make a further modification in our scheme. Namely, the master key generation algorithm first selects two strings str_p, str_q of length l, then computes the generators as $P = H_1(str_p),\ Q = H_1(str_q)$. In the security proof (in the random oracle model), it is straightforward to embed the CDH problem instance into the master public key mpk as $H_1(str_p) = P,\ H_1(str_q) = aP$ by programming the random oracle H_1.

3.3. An example of fully chosen-key attacks

For simplicity, we assume there are two users with identities $\mathsf{ID}_1, \mathsf{ID}_2$ and public keys $pk_{\mathsf{ID}_1}, pk_{\mathsf{ID}_2}$. The adversary $\mathcal{A}$ holds their private keys $sk_{\mathsf{ID}_1} = (d_{\mathsf{ID}_1}, \nu_{\mathsf{ID}_1}), sk_{\mathsf{ID}_2} = (d_{\mathsf{ID}_2}, \nu_{\mathsf{ID}_2})$ and aims to output two invalid single signatures σ_1, σ_2 on messages m_1, m_2 signed by users $\mathsf{ID}_1, \mathsf{ID}_2$, respectively that can be aggregated into a valid aggregate signature Σ. It works as follows:

1. For any messages m_1, m_2, select random $r_i \in \mathbb{Z}_q$, computes $U_i = r_i \cdot P$ and $h_i = H_3(m_i, \mathsf{ID}_i, pk_{\mathsf{ID}_i}, U_i)$, for $i = 1, 2$.
2. Pick a random $R \in \mathbb{G}$ and compute

$$V_1 = d_{\mathsf{ID}_1} + h_1 \cdot r_1 \cdot P_{pub} + h_1 \cdot \upsilon_{\mathsf{ID}_1} \cdot H_1(Q) + r_1 \cdot H_1(Q) + R$$
$$V_2 = d_{\mathsf{ID}_2} + h_2 \cdot r_2 \cdot P_{pub} + h_2 \cdot \upsilon_{\mathsf{ID}_2} \cdot H_1(Q) + r_2 \cdot H_1(Q) - R$$

3. Compute $V = V_1 + V_2$ and output $\sigma_1 = (U_1, V_1), \sigma_2 = (U_2, V_2)$ as single signatures and $\Sigma = (U_1, U_2, V)$ as the aggregate signature.

It is easy to see that the aggregate signature $\Sigma = (U_1, U_2, V)$ is generated from single signatures σ_1, σ_2. In addition, V_1, V_2 in single signatures σ_1, σ_2 are computed following the signing algorithm with the users' private keys and the random element R. Since R in V_1 and $-R$ in V_2 cancel out after aggregation as $V = V_1 + V_2$ and hence the aggregate signature Σ is valid. On the other hand, both the single signatures σ_1, σ_2 are invalid because of the random element R in V_1 and V_2, respectively.

Note that such an attack does make sense in practical applications. For example, two dishonest users ID_1 and ID_2 with public keys pk_1 and pk_2 want to cheat the verifier. They cooperatively work as follows:

1. ID_1 first selects random $r_1 \in \mathbb{Z}_q, R_1 \in \mathbb{G}$ and compute $T_1 = r_1 \cdot H_1(Q) + R_1$. Then, ID_1 delivers T_1 to ID_2.
2. ID_2 selects random $r_2 \in \mathbb{Z}_q, R_2 \in \mathbb{G}$ and computes $T_2 = r_2 \cdot H_1(Q) + R_2$. Then, ID_2 delivers T_2 to ID_1.
3. After the exchange, ID_1 computes $U_1 = r_1 \cdot P, h_1 = H_3(m_1, \mathsf{ID}_1, pk_{\mathsf{ID}_1}, U_1)$. Next, ID_1 computes V_1 with its private key $sk_{\mathsf{ID}_1} = (d_{\mathsf{ID}_1}, u_{\mathsf{ID}_1})$, where

$$V_1 = d_{\mathsf{ID}_1} + h_1 \cdot r_1 \cdot P_{pub} + h_1 \cdot u_{\mathsf{ID}_1} \cdot H_1(Q) + T_2 - R_1.$$

   Similarly, ID_2 computes $U_2 = r_2 \cdot P, h_2 = H_3(m_2, \mathsf{ID}_2, pk_{\mathsf{ID}_2}, U_2)$. Next, ID_2 computes V_2 with its private key $sk_{\mathsf{ID}_2} = (d_{\mathsf{ID}_2}, u_{\mathsf{ID}_2})$, where

$$V_2 = d_{\mathsf{ID}_2} + h_2 \cdot r_2 \cdot P_{pub} + h_2 \cdot u_{\mathsf{ID}_2} \cdot H_1(Q) + T_1 - R_2.$$

4. At last, ID_1 and ID_2 output $\sigma_1 = (U_1, V_1)$ and $\sigma_2 = (U_2, V_2)$ as signatures on messages m_1 and m_2 respectively.

The aggregate signature $\Sigma = (U_1, U_2, V_1 + V_2)$. It is easy to see that Σ is valid since the random elements R_1, R_2 cancel out after aggregation. However, both the two single signatures σ_1, σ_2 are invalid.

4. An improved CLAS scheme secure against fully chosen-key attacks

In this section, we show how to reinforce the scheme in the last section to withstand the fully chosen-key attacks. The single signatures achieve *existential unforgeability against chosen-message attacks* and the aggregate signatures achieve *equivalent validity against fully chosen-key attacks*.

4.1. Scheme construction

- $\mathsf{Setup}(1^\lambda)$: The KGC runs the master key generation algorithm with the security parameter λ as input. It chooses pairing groups $\mathbb{PG} = (\mathbb{G}, \mathbb{G}_T, q, e)$ and three hash functions $H_1, H_2 : \{0, 1\}^* \to \mathbb{G}$ and $H_3, H_4 : \{0, 1\}^* \to \mathbb{Z}_q$. Then, it randomly picks two bit strings str_p, str_q of length l and computes $P = H_1(str_p), Q = H_1(str_q)$. Next, it selects a random $s \in \mathbb{Z}_q$ and computes $P_{pub} = sP$. It outputs the master public key $mpk = (\mathbb{PG}, l, str_p, str_q, P, Q, P_{pub}, H_1, H_2, H_3, H_4)$ and keeps the master secret key $msk = s$ secretly. After the master public key mpk has been published, the intended verifier selects a random $y \in \mathbb{Z}_q$ as the secret verification key $sk_{\mathsf{ver}} = y$, computes, and publishes the public verification key $pk_{\mathsf{ver}} = yP$.
- $\mathsf{PartialKeyGen}(mpk, msk, \mathsf{ID})$: The KGC runs the partial private key generation algorithm with the master public key mpk, master secret key $msk = s$, and a user's identity ID as input. It computes $H_2(\mathsf{ID}) \in \mathbb{G}$ and $d_{\mathsf{ID}} = s \cdot H_2(\mathsf{ID})$. Then, it outputs d_{ID} as the partial private key of user ID.
- $\mathsf{UserKeyGen}(mpk, \mathsf{ID})$: The user with identity ID runs the user key generation algorithm with the master public key mpk and identity ID as input. It selects a random $x \in \mathbb{Z}_q$ and computes $pk_{\mathsf{ID}} = xP$. It sets the secret value $\upsilon_{\mathsf{ID}} = x$ and outputs $(pk_{\mathsf{ID}}, \upsilon_{\mathsf{ID}})$ as a pair of public key and secret value.
- $\mathsf{Sign}(mpk, sk_{\mathsf{ID}}, m)$: The user with identity ID runs the signing algorithm with the master public key mpk, private key $sk_{\mathsf{ID}} = (d_{\mathsf{ID}}, \upsilon_{\mathsf{ID}})$, and a message m as input. It performs as follows:
  1. Choose a random element $r \in \mathbb{Z}_q$ and compute $U = rP$.
  2. Compute $h = H_3(m, \mathsf{ID}, pk_{\mathsf{ID}}, U) \in \mathbb{Z}_q$ and

$$V = d_{\mathsf{ID}} + h \cdot r \cdot P_{pub} + h \cdot \upsilon_{\mathsf{ID}} \cdot Q + r \cdot Q.$$

  3. Set $\sigma = (U, V)$.

It outputs σ as a signature on message m signed by user ID.

- $\mathsf{Verify}(mpk, \mathsf{ID}, pk_{\mathsf{ID}}, m, \sigma)$: Any user can run the verification algorithm with the master public key mpk, identity ID, public key pk_{ID}, message m, and a signature $\sigma = (U, V)$ as input. It first computes $h = H_3(m, \mathsf{ID}, pk_{\mathsf{ID}}, U)$ and then checks whether the equation

$$e(V, P) = e\Big(H_2(\mathsf{ID}) + h \cdot U, P_{pub}\Big) \cdot e(h \cdot pk_{\mathsf{ID}} + U, Q)$$

holds. If yes, it outputs "Accept". Otherwise, it outputs "Reject".

- $\mathsf{Aggregate}(mpk, \{(\mathsf{ID}_i, pk_{\mathsf{ID}_i}, m_i, \sigma_i) \mid i = 1, \ldots, n\}, pk_{\mathsf{ver}})$: Any user can run the aggregation algorithm with the master public key mpk, a set of four-tuples $(\mathsf{ID}_i, pk_{\mathsf{ID}_i}, m_i, \sigma_i)$ consisting of identities, public keys, message, signatures, where $\sigma_i = (U_i, V_i)$, and the public verification key $pk_{\mathsf{ver}} = yP$ as input. It first computes $V = \sum_{i=1}^{n} V_i$ and $\mu = H_4(e(V_1, pk_{\mathsf{ver}}) \parallel \ldots \parallel e(V_n, pk_{\mathsf{ver}}))$. It then outputs $\Sigma = (U_1, \ldots, U_n, V, \mu)$ as the aggregate signature on messages signed by the identities and public keys in the set of four-tuples.
- $\mathsf{AggVer}(mpk, \{(\mathsf{ID}_i, pk_{\mathsf{ID}_i}, m_i) \mid i = 1, \ldots, n\}, \Sigma, sk_{\mathsf{ver}})$: The intend verifier runs the aggregation verification algorithm with the master public key mpk, a set of trituples $(\mathsf{ID}_i, pk_{\mathsf{ID}_i}, m_i)$ consisting of identities, public keys, messages, an aggregate signature $\Sigma = (U_1, \ldots, U_n, V, \mu)$, and the secret verification key $sk_{\mathsf{ver}} = y$ as input. It first computes $h_i = H_3(m_i, \mathsf{ID}_i, pk_{\mathsf{ID}_i}, U_i)$, for $i = 1, \ldots, n$ and then checks whether the following equations hold

$$e(V, P) = e\left(\sum_{i=1}^{n}\Big(H_2(\mathsf{ID}_i) + h_i \cdot U_i\Big), P_{pub}\right) \cdot e\left(\sum_{i=1}^{n}\Big(h_i \cdot pk_{\mathsf{ID}_i} + U_i\Big), Q\right),$$

$$\mu = H_4\left(\begin{array}{c} e\Big(H_2(\mathsf{ID}_1) + h_1 \cdot U_1, y \cdot P_{pub}\Big) \cdot e(h_1 \cdot pk_{\mathsf{ID}_1} + U_1, y \cdot Q) \parallel \ldots \\ \parallel e\Big(H_2(\mathsf{ID}_n) + h_n \cdot U_n, y \cdot P_{pub}\Big) \cdot e(h_n \cdot pk_{\mathsf{ID}_n} + U_n, y \cdot Q) \end{array}\right).$$

If yes, it outputs "Accept". Otherwise, it outputs "Reject".

4.2. Security proof

Theorem 1. *The single signature of the proposed certificateless aggregate signature (CLAS) scheme is existentially unforgeable against chosen-message attacks by a strong type I adversary and a super type II (malicious KGC) adversary, assuming that the CDH assumption holds in group $\mathbb{G}$.*

Proof. Theorem 1 follows readily from Lemmas 1 and 2 given below, where the basic idea of the proofs is similar to [5]. In our construction, instead of selecting a random generator $P \in \mathbb{G}$, we picks a random string str_p and computes generator $P = H_1(str_p)$. In the security proof, the simulator only needs to program the random oracle of the hash function H_1 to embed the problem instance into the responses to hash queries of strings str_p, str_q. □

Lemma 1. *Suppose H_1, H_2, H_3 are random oracles, where the adversary makes at most q_H queries on H_2. If there exists a strong type I adversary $\mathcal{A}_{\mathsf{I}}$ who has advantage ϵ in breaking the proposed CLAS scheme, then we can construct a simulator $\mathcal{B}_{\mathsf{I}}$ that has advantage $\frac{\epsilon}{q_H}$ in breaking the CDH assumption in group $\mathbb{G}$.*

Proof of Lemma 1. Suppose the simulator $\mathcal{B}_{\mathsf{I}}$ receives a CDH instance (P, aP, bP) from pairing groups $\mathbb{PG}$ and its goal is to compute abP. It controls the random oracles H_1, H_2, H_3 and interacts with $\mathcal{A}_{\mathsf{I}}$ as follows.

Setup. $\mathcal{B}_{\mathsf{I}}$ sets the pairing groups $\mathbb{PG}$ as the problem instance it receives. Then, it picks two random bit strings str_p, str_q of length l, a random $t \in \mathbb{Z}_q$, sets $Q = tP$, and records two tuples (str_p, P), (str_q, Q) for hash queries on random oracle H_1. Next, it sets $P_{pub} = aP$ from the problem instance and outputs the master public key

$$mpk = (\mathbb{PG}, l, str_p, str_q, P, Q, P_{pub}).$$

Note that the master secret key msk is equal to the discrete logarithm a from the problem instance, which is unknown to $\mathcal{B}_{\mathsf{I}}$.

Hash Query to H_1. $\mathcal{B}_{\mathsf{I}}$ maintains an initially empty list $\mathcal{L}_1$ storing tuples of structure (X, R_X), where X and R_X refer to the hash query and hash value returned to $\mathcal{A}_{\mathsf{I}}$ respectively. After the **Setup** phase, two tuples (str_p, P), (str_q, Q) are inserted into $\mathcal{L}_1$. On receiving a hash query on X from $\mathcal{A}_{\mathsf{I}}$, it first checks whether X exists in $\mathcal{L}_1$. If yes, it responds according to the tuple stored. Otherwise, it picks a random $R_X \in \mathbb{G}$, returns R_X to $\mathcal{A}_{\mathsf{I}}$, and inserts the tuple (X, R_X) into $\mathcal{L}_1$.

Hash Query to H_2. $\mathcal{B}_{\mathsf{I}}$ maintains an initially empty list $\mathcal{L}_2$ storing tuples of structure (X, s_X, S_X), where X, s_X, and S_X refer to the hash query, secret information used to compute the hash value, and hash value returned to $\mathcal{A}_{\mathsf{I}}$ respectively. Suppose $\mathcal{A}_{\mathsf{I}}$ makes at most q_H queries to H_2, $\mathcal{B}_{\mathsf{I}}$ randomly picks a $j^* \in [1, q_H]$. On receiving the jth hash query on $X = \mathsf{ID}_j$ that has not been queried before, $\mathcal{B}_{\mathsf{I}}$ selects a random $s_X \in \mathbb{Z}_q$ and works as follows:

- If $j = j^*$, it sets $S_X = bP$ from the problem instance.
- Otherwise, it sets $S_X = s_X \cdot P$.

$\mathcal{B}_I$ returns S_X to $\mathcal{A}_I$ and inserts (X, s_X, S_X) into $\mathcal{L}_2$.

Hash query to H$_3$. $\mathcal{B}_I$ maintains an initially empty list $\mathcal{L}_3$ storing tuples of structure (X, h_X), where X and h_X refer to the hash query and hash value returned to $\mathcal{A}_I$ respectively. On receiving a hash query on X that has not been queried before, $\mathcal{B}_I$ selects a random $h_X \in \mathbb{Z}_q$, returns h_X to $\mathcal{A}_I$, and inserts (X, h_X) into $\mathcal{L}_3$.

Partial private key query. On receiving a partial private key query on identity ID_j, $\mathcal{B}_I$ first checks whether $j = j^*$. If yes, it aborts. Otherwise, it checks whether ID_j has been queried to oracle H_2. If no, $\mathcal{B}_I$ makes a hash query on ID_j to oracle H_2 by itself. Then, it searches the tuple that contains ID_j and records s_{ID_j}. Next, it computes $d_{\mathsf{ID}_j} = s_{\mathsf{ID}_j} \cdot P_{pub}$ as the partial private key of user ID_j and returns d_{ID_j} to $\mathcal{A}_I$.

Public key query. $\mathcal{B}_I$ maintains an initially empty list $\mathcal{L}_{pk}$ storing tuples of structure $(\mathsf{ID}, pk_{\mathsf{ID}}, v_{\mathsf{ID}}, pk'_{\mathsf{ID}}, v'_{\mathsf{ID}})$. On receiving a public key query on identity ID_j, $\mathcal{B}_I$ searches the tuple that contains ID_j in $\mathcal{L}_{pk}$ and works as follows:

- If the tuple has not been generated yet, it runs the user key generation algorithm $\mathsf{UserKeyGen}$ to generate $(pk_{\mathsf{ID}_j}, v_{\mathsf{ID}_j})$, returns pk_{ID_j} to $\mathcal{A}_I$, and inserts $(\mathsf{ID}, pk_{\mathsf{ID}_j}, v_{\mathsf{ID}_j}, -, -)$ into $\mathcal{L}_{pk}$.
- If the tuple has been generated and the public key pk_{ID_j}has not been replaced by $\mathcal{A}_I$, $\mathcal{B}_I$ returns pk_{ID_j} to $\mathcal{A}_I$.
- If the tuple has been generated and the public key pk_{ID_j} has been replaced by pk'_{ID_j}, $\mathcal{B}_I$ returns pk'_{ID_j} to $\mathcal{A}_I$.

Secret value query. On receiving a secret value query on identity ID_j, $\mathcal{B}_I$ searches the tuple that contains identity ID_j in $\mathcal{L}_{pk}$ and works as follows:

- If the tuple has not been generated yet, it runs the user key generation algorithm $\mathsf{UserKeyGen}$ to generate $(pk_{\mathsf{ID}_j}, v_{\mathsf{ID}_j})$, returns v_{ID_j} to $\mathcal{A}_I$, and inserts $(\mathsf{ID}, pk_{\mathsf{ID}_j}, v_{\mathsf{ID}_j}, -, -)$ into $\mathcal{L}_{pk}$.
- If the tuple has been generated, $\mathcal{B}_I$ returns v_{ID_j} to $\mathcal{A}_I$ no matter whether the public key pk_{ID_j} has been replaced or not.

Public key replace query. On receiving a public key replace query on identity ID_j, $\mathcal{B}_I$ records the public key and secret value pair $(pk'_{\mathsf{ID}_j}, v'_{\mathsf{ID}_j})$ submitted by $\mathcal{A}_I$, searches the tuple that contains the identity ID_j in $\mathcal{L}_{pk}$ and works as follows:

- If the tuple has not been generated yet, it runs the user key generation algorithm $\mathsf{UserKeyGen}$ to generate the public key and secret value pair $(pk_{\mathsf{ID}_j}, v_{\mathsf{ID}_j})$ and inserts $(\mathsf{ID}, pk_{\mathsf{ID}_j}, v_{\mathsf{ID}_j}, pk'_{\mathsf{ID}_j}, v'_{\mathsf{ID}_j})$ into $\mathcal{L}_{pk}$.
- If the tuple has been generated, $\mathcal{B}_I$ updates the tuple by
$$(\mathsf{ID}, pk_{\mathsf{ID}_j}, v_{\mathsf{ID}_j}, pk'_{\mathsf{ID}_j}, v'_{\mathsf{ID}_j}).$$

Signing query. On receiving a signing query on message m signed by identity ID_j, $\mathcal{B}_I$ searches $\mathcal{L}_{pk}$ for the tuple containing ID_j and works as follows:

- If $j \neq j^*$, $\mathcal{B}_I$ first computes the partial private key d_{ID_j}. Then, it determines the secret value as follows:
  - If the tuple has not been generated, $\mathcal{B}_I$ runs the user key generation algorithm $\mathsf{UserKeyGen}$ to generate $(pk_{\mathsf{ID}_j}, v_{\mathsf{ID}_j})$ and inserts the tuple $(\mathsf{ID}, pk_{\mathsf{ID}_j}, v_{\mathsf{ID}_j}, -, -)$ into $\mathcal{L}_{pk}$. Then, it sets $sk_{\mathsf{ID}_j} = (d_{\mathsf{ID}_j}, v_{\mathsf{ID}_j})$ as the private key of the user with identity ID_j.
  - If the tuple has been generated and the secret value v_{ID_j} has not been replaced, $\mathcal{B}_I$ sets $sk_{\mathsf{ID}_j} = (d_{\mathsf{ID}_j}, v_{\mathsf{ID}_j})$ as the private key.
  - If the tuple has been generated and the secret value v_{ID_j} has been replaced by v'_{ID_j}, $\mathcal{B}_I$ sets $sk_{\mathsf{ID}_j} = (d_{\mathsf{ID}_j}, v'_{\mathsf{ID}_j})$ as the private key.

  Next, it runs the signing algorithm Sign with the private key sk_{ID_j} as input to generate the signature σ and returns σ to $\mathcal{A}_I$.
- If $j = j^*$, $\mathcal{B}_I$ selects random $r, h \in \mathbb{Z}_q$ and computes $U = rP - h^{-1} \cdot H_2(\mathsf{ID}_j)$, $V = h \cdot r \cdot P_{pub} + h \cdot v \cdot Q + rQ + h^{-1} \cdot t \cdot bP$. Then, $\mathcal{B}_I$ works as follows:
  - If the public key pk_{ID_j} has not been replaced, it inserts the tuple (X, h) into $\mathcal{L}_3$, where $X = (m, \mathsf{ID}_j, pk_{\mathsf{ID}_j}, U)$.
  - If the public key pk_{ID_j} has been replaced by pk_{ID_j}, it inserts the tuple (X, h) into $\mathcal{L}_3$, where $X = (m, \mathsf{ID}_j, pk'_{\mathsf{ID}_j}, U)$.

  $\mathcal{B}_I$ sets $\sigma = (U, V)$ as the signature on message m signed by user ID_j and returns σ to $\mathcal{A}_I$. Note that σ is valid since after programming the random oracle H_3, where $h = H_3(m, \mathsf{ID}_j, pk_{\mathsf{ID}_j}, U)$, it holds that
$$\begin{aligned}
&e(V, P)\\
&= e(h \cdot rP_{pub} + h \cdot v \cdot Q + rQ - h^{-1} \cdot t \cdot bP, P)\\
&= e(h \cdot rP, P_{pub}) \cdot e\Big((h \cdot v + r) \cdot P, Q\Big) \cdot e(h^{-1} \cdot bP, tP)\\
&= e\Big(h \cdot \big(U + h^{-1} \cdot H_2(\mathsf{ID}_j)\big), P_{pub}\Big) \cdot e\Big(h \cdot pk_{\mathsf{ID}_j} + rP - h^{-1} \cdot H_2(\mathsf{ID}_j), Q\Big)\\
&= e\Big(H_2(\mathsf{ID}_j) + h \cdot U, P_{pub}\Big) \cdot e(h \cdot pk_{\mathsf{ID}_j} + U, Q)
\end{aligned}$$

Forgery. At last, $\mathcal{A}_I$ outputs its forgery of a signature on message m under identity ID_j and public key pk_{ID_j}. Applying the forking lemma [17], $\mathcal{A}_I$ outputs two valid signatures $\sigma_1 = (U, V_1), \sigma_2 = (U, V_2)$ if replaying with the same random tape to oracle H_3. Specifically, h_1, h_2 are two hash values returned to $\mathcal{A}_I$ as responses of hash query on $(m, \mathsf{ID}_j, pk_{\mathsf{ID}_j}, U)$.

If the identity ID_j of $\mathcal{A}_I$'s forgery does not satisfy $j = j^*$, $\mathcal{B}_I$ aborts. Otherwise, it computes $\frac{h_1 \cdot V_2 - h_2 \cdot V_1 - (h_1 - h_2) \cdot t \cdot U}{h_1 - h_2}$ as the solution to the CDH problem instance. For valid signatures $\sigma_1 = (U, V_1), \sigma_2 = (U, V_2)$, we have

$$e(V_1, P) = e\Big(H_2(\mathsf{ID}_j) + h_1 \cdot U, P_{pub}\Big) \cdot e(h_1 \cdot pk_{\mathsf{ID}_j} + U, Q) \quad (1)$$

$$e(V_2, P) = e\Big(H_2(\mathsf{ID}_j) + h_2 \cdot U, P_{pub}\Big) \cdot e(h_2 \cdot pk_{\mathsf{ID}_j} + U, Q) \quad (2)$$

The correctness of the solution follows readily from Eqs. (1) and (2).

$$\begin{aligned} & e\left(\frac{h_1 \cdot V_2 - h_2 \cdot V_1 - (h_1 - h_2) \cdot t \cdot U}{h_1 - h_2}, P\right) \\ &= \left(\frac{e(h_1 \cdot V_2, P_{pub})}{e(h_2 \cdot V_1, P_{pub}) \cdot e\Big((h_1 - h_2) \cdot t \cdot U, P\Big)}\right)^{\frac{1}{h_1 - h_2}} \\ &= \left(\frac{e\Big(h_1 \cdot H_2(\mathsf{ID}_j) + h_1 h_2 U, P_{pub}\Big) \cdot e(h_1 h_2 pk_{\mathsf{ID}_j} + h_1 U, Q)}{e\Big(h_2 \cdot H_2(\mathsf{ID}_j) + h_2 h_1 U, P_{pub}\Big) \cdot e(h_2 h_1 pk_{\mathsf{ID}_j} + h_2 U, Q) \cdot e\Big((h_1 - h_2)U, tP\Big)}\right)^{\frac{1}{h_1 - h_2}} \\ &= \left(e\Big((h_1 - h_2) \cdot H_2(\mathsf{ID}_j), P_{pub}\Big)\right)^{\frac{1}{h_1 - h_2}} \\ &= e\Big(H_2(\mathsf{ID}_j), P_{pub}\Big) = e(bP, aP). \end{aligned}$$

This completes the description of the simulation and solution. Next, we analyze the advantage of $\mathcal{B}_I$. Whether it can compute the solution depends on the following events.

1. $\mathcal{B}_I$ does not abort during the partial private key query phase.
2. $\mathcal{B}_I$ does not abort during the forgery phase.
3. $\mathcal{A}_I$ outputs a valid forgery.

$\mathcal{A}_I$ makes at most q_H hash queries to oracle H_2 and $\mathcal{B}_I$ randomly select a $j^* \in [1, q_H]$. Therefore, the identity ID_j in $\mathcal{A}_I$'s forgery satisfies $j = j^*$ with probability at least $\frac{1}{q_H}$. According to the security model, $\mathcal{A}_I$ does not make the partial private key query on the identity ID_j of its forgery. As a result, $\mathcal{B}_I$ does not abort during the simulation with probability at least $\frac{1}{q_H}$. Since $\mathcal{A}_I$ has advantage ϵ in forging a valid signature, $\mathcal{B}_I$ has advantage $\frac{\epsilon}{q_H}$ in solving the CDH problem instance. This completes the proof of Lemma 1. □

Lemma 2. *Suppose H_1, H_2, H_3 are random oracles, where the adversary makes at most q_H queries on H_2. If there exists a super type II (malicious KGC) adversary $\mathcal{A}_{II}$ who has advantage ϵ in breaking the proposed CLAS scheme, then we can construct a simulator $\mathcal{B}_{II}$ that has advantage $\frac{\epsilon}{q_H}$ in breaking the CDH assumption in group $\mathbb{G}$.*

Proof of Lemma 1. Proof of Lemma 2. Suppose the adversary $\mathcal{A}_{II}$ publishes pairing groups $\mathbb{PG}$, two bit strings str_p, str_q of length l, element P_{pub}, and master secret key $s \in \mathbb{Z}_q$. The simulator $\mathcal{B}_{II}$ receives a CDH instance (P, aP, bP) and its goal is to compute abP. It controls the random oracles H_1, H_2, H_3 and interacts with $\mathcal{A}_{II}$ as follows.

Setup. $\mathcal{B}_{II}$ uses the problem instance (P, aP, bP), sets $Q = aP$, and programs the random oracle H_1, such that the responses to hash queries on strings str_p, str_q as P, Q respectively. The master public key includes $mpk = (\mathbb{PG}, l, str_p, str_q, P, Q, P_{pub})$. The master secret key is published by $\mathcal{A}_{II}$.

Hash Query to H$_1$. $\mathcal{B}_{II}$ maintains a list $\mathcal{L}_1$ storing tuples of structure (X, R_X), where X and R_X refer to the hash query and hash value returned to $\mathcal{A}_{II}$ respectively. After $\mathcal{A}_{II}$ publishing strings str_p, str_q, $\mathcal{B}_{II}$ inserts two tuples (str_p, P), (str_q, Q) into $\mathcal{L}_1$. On receiving a hash query on X from $\mathcal{A}_{II}$, it first checks whether X exists in $\mathcal{L}_1$. If yes, it responds according to the tuple stored. Otherwise, it picks a random $R_X \in \mathbb{G}$, returns R_X to $\mathcal{A}_I$, and inserts the tuple (X, R_X) into $\mathcal{L}_1$.

Hash Query to H$_2$. $\mathcal{B}_{II}$ maintains an initially empty list $\mathcal{L}_2$ storing tuples of structure (X, s_X, S_X), where X, s_X, and S_X refer to the hash query, secret information used to compute the hash value, and the hash value returned to $\mathcal{A}_{II}$, respectively. On receiving a hash query on $X = \mathsf{ID}_j$ that has not been queried before, $\mathcal{B}_{II}$ selects a random $s_X \in \mathbb{Z}_q$, sets $S_X = s_X P$, returns S_X to $\mathcal{A}_{II}$, and inserts (X, s_X, S_X) into $\mathcal{L}_2$.

Hash Query to H$_3$. $\mathcal{B}_{II}$ maintains an initially empty list $\mathcal{L}_3$ storing tuples of structure (X, h_X), where X and h_X refer to the hash query and hash value returned to $\mathcal{A}_{II}$ respectively. On receiving a hash query on X that has not been queried before, $\mathcal{B}_{II}$ selects a random $h_X \in \mathbb{Z}_q$, returns h_X to $\mathcal{A}_{II}$, and inserts (X, h_X) into $\mathcal{L}_3$.

Public key query $\mathcal{B}_I$ maintains an initially empty list $\mathcal{L}_{pk}$ storing tuples of structure $(\mathsf{ID}, pk_{\mathsf{ID}}, v_{\mathsf{ID}}, pk'_{\mathsf{ID}})$. Suppose $\mathcal{A}_{II}$ makes at most q_H queries to oracle H_2, $\mathcal{B}_{II}$ randomly picks a $j^* \in [1, q_H]$. On receiving a public key query on identity ID_j, $\mathcal{B}_I$ searches the tuple that contains the identity ID_j in $\mathcal{L}_{pk}$ and works as follows:

- If the tuple has not been generated, it selects a random $s_j \in \mathbb{Z}_q$ and checks whether $j = j^*$. If yes, $\mathcal{B}_{II}$ sets $pk_{\mathsf{ID}_j} = bP$, returns pk_{ID_j} to $\mathcal{A}_{II}$, and inserts $(\mathsf{ID}, pk_{\mathsf{ID}_j}, -, -)$ into $\mathcal{L}_{pk}$. Otherwise, it computes $pk_{\mathsf{ID}_j} = s_jP$, returns pk_{ID_j} to $\mathcal{A}_{II}$, and inserts $(\mathsf{ID}, pk_{\mathsf{ID}_j}, s_j, -)$ into $\mathcal{L}_{pk}$.
- If the tuple has been generated and the public key pk_{ID_j} has not been replaced, $\mathcal{B}_I$ returns pk_{ID_j} to $\mathcal{A}_{II}$.
- If the tuple has been generated and the public key pk_{ID_j} has been replaced by pk'_{ID_j}, $\mathcal{B}_{II}$ returns pk'_{ID_j} to $\mathcal{A}_{II}$.

Secret value query. On receiving a secret value query on identity ID_j, $\mathcal{B}_{II}$ searches the tuple that contains identity ID_j in $\mathcal{L}_{pk}$ and works as follows:

- If $j = j^*$, it aborts.
- If $j \neq j^*$ and the tuple has not been generated, it selects a random $s_j \in \mathbb{Z}_q$ and sets the secret value $v_{\mathsf{ID}_j} = s_j$. Then, it computes $pk_{\mathsf{ID}_j} = s_jP$, returns s_j to $\mathcal{A}_{II}$, and inserts $(\mathsf{ID}, pk_{\mathsf{ID}_j}, v_{\mathsf{ID}_j}, -)$ into $\mathcal{L}_{pk}$.
- If $j \neq j^*$ and the tuple has been generated, $\mathcal{B}_{II}$ returns v_{ID_j} to $\mathcal{A}_{II}$ no matter whether the public key pk_{ID_j} has been replaced or not.

Public key replace query. On receiving a public key replace query on identity ID_j, $\mathcal{B}_{II}$ records the public key pk'_{ID_j} submitted by $\mathcal{A}_{II}$, searches the tuple that contains identity ID_j in $\mathcal{L}_{pk}$, and works as follows:

- If $j = j^*$, it aborts.
- If $j \neq j^*$ and the tuple has not been generated, it inserts $(\mathsf{ID}, -, -, pk'_{\mathsf{ID}_j})$ into $\mathcal{L}_{pk}$.
- If $j \neq j^*$ and the tuple has been generated, $\mathcal{B}_{II}$ updates the tuple by

$$(\mathsf{ID}, pk_{\mathsf{ID}_j}, v_{\mathsf{ID}_j}, pk'_{\mathsf{ID}_j}).$$

Signing query. On receiving a signing query on message m signed by identity ID_j, $\mathcal{B}_{II}$ first searches $\mathcal{L}_{pk}$ to find the secret value v_{ID_j} of user ID_j. Then, it computes the partial private key d_{ID_j} of user ID_j with the master secret key s, selects random $h, t \in \mathbb{Z}_q$ and works as follows:

- If $v_{\mathsf{ID}_j} \neq -$, i.e. $\mathcal{B}_{II}$ holds the secret value, it runs the signing algorithm Sing with the private key $sk_{\mathsf{ID}_j} = (d_{\mathsf{ID}_j}, v_{\mathsf{ID}_j})$ as input to generate the signature σ on message m signed by user ID_j.
- If $v_{\mathsf{ID}_j} = -$ and the public key pk_{ID_j} has not been replaced, it computes $U = t \cdot P_{pub} - h \cdot pk_{\mathsf{ID}_j}, V = d_{\mathsf{ID}_j} + h \cdot s \cdot U + t \cdot s \cdot Q$ and inserts the tuple (X, h) into $\mathcal{L}_3$, where $X = (m, \mathsf{ID}_j, pk_{\mathsf{ID}_j}, U)$.
- If $v_{\mathsf{ID}_j} = -$ and the public key pk_{ID_j} has been replaced by pk'_{ID_j}, it computes $U = t \cdot P_{pub} - h \cdot pk'_{\mathsf{ID}_j}, V = d_{\mathsf{ID}_j} + h \cdot s \cdot U + t \cdot s \cdot Q$ and inserts the tuple (X, h) into $\mathcal{L}_3$, where $X = (m, \mathsf{ID}_j, pk'_{\mathsf{ID}_j}, U)$.

$\mathcal{B}_{II}$ sets $\sigma = (U, V)$ as the signature on message m signed by user ID_j and returns σ to $\mathcal{A}_{II}$. Note that σ is valid since after programming the random oracle H_3, where $h = H_3(m, \mathsf{ID}_j, pk_{\mathsf{ID}_j}, U)$, it holds that

$$\begin{aligned} e(V, P) &= e(d_{\mathsf{ID}_j} + h \cdot s \cdot U + t \cdot s \cdot Q, P) \\ &= e(d_{\mathsf{ID}_j} + h \cdot U, sP) \cdot e(tQ, sP) \\ &= e\Big(H_2(\mathsf{ID}_j) + h \cdot U, P_{pub}\Big) \cdot e(t \cdot P_{pub}, Q) \\ &= e\Big(H_2(\mathsf{ID}_j) + h \cdot U, P_{pub}\Big) \cdot e(h \cdot pk_{\mathsf{ID}_j} + U, Q) \end{aligned}$$

Forgery. At last, $\mathcal{A}_{II}$ outputs its forgery of a signature on message m under identity ID_j and public key pk_{ID_j}. Applying the forking lemma [17], $\mathcal{A}_{II}$ outputs two valid signatures $\sigma_1 = (U, V_1), \sigma_2 = (U, V_2)$ if replaying with the same random tape to oracle H_3. Specifically, h_1, h_2 are two hash values returned to $\mathcal{A}_{II}$ as responses of hash query on $(m, \mathsf{ID}_j, pk_{\mathsf{ID}_j}, U)$.

If the identity ID_j of $\mathcal{A}_{II}$'s forgery does not satisfy $j = j^*$, $\mathcal{B}_{II}$ aborts. Otherwise, it computes $\frac{V_1 - V_2 - (h_1 - h_2) \cdot s \cdot U}{h_1 - h_2}$ as the solution to the CDH problem instance. For valid signatures $\sigma_1 = (U, V_1), \sigma_2 = (U, V_2)$, we have

$$e(V_1, P) = e\Big(H_2(\mathsf{ID}_j) + h_1 \cdot U, P_{pub}\Big) \cdot e(h_1 \cdot pk_{\mathsf{ID}_j} + U, Q) \tag{3}$$

$$e(V_2, P) = e\Big(H_2(\mathsf{ID}_j) + h_2 \cdot U, P_{pub}\Big) \cdot e(h_2 \cdot pk_{\mathsf{ID}_j} + U, Q) \tag{4}$$

The correctness of the solution follows readily from Eqs. (3) and (4).

$$e\left(\frac{V_1 - V_2 - (h_1 - h_2) \cdot s \cdot U}{h_1 - h_2}, P\right) = \left(\frac{e(V_1, P)}{e(V_2, P) \cdot e\Big((h_1 - h_2) \cdot s \cdot U, P\Big)}\right)^{\frac{1}{h_1 - h_2}}$$

$$= \left(\frac{e\Big(H_2(\mathsf{ID}_j) + h_1 \cdot U, P_{pub}\Big) \cdot e(h_1 \cdot pk_{\mathsf{ID}_j} + U, Q)}{e\Big(H_2(\mathsf{ID}_j) + h_2 \cdot U, P_{pub}\Big) \cdot e(h_2 \cdot pk_{\mathsf{ID}_j} + U, Q) \cdot e\Big((h_1 - h_2) \cdot s \cdot U, P\Big)} \right)^{\frac{1}{h_1 - h_2}}$$

$$= \left(\frac{e\Big(h_1 - h_2) \cdot U, P_{pub}\Big) \cdot e\Big((h_1 - h_2) \cdot pk_{\mathsf{ID}_j}, Q\Big)}{e\Big((h_1 - h_2) \cdot U, sP\Big)} \right)^{\frac{1}{h_1 - h_2}}$$

$$= e(pk_{\mathsf{ID}_j}, Q) = e(bP, aP).$$

This completes the description of the simulation and solution. Next, we analyze the advantage of $\mathcal{B}_{\text{II}}$. Whether it can compute the solution depends on the following events.

1. $\mathcal{B}_{\text{I}}$ does not abort during the secret value query phase.
2. $\mathcal{B}_{\text{I}}$ does not abort during the public key replace query phase.
3. $\mathcal{B}_{\text{I}}$ does not abort during the forgery phase.
4. $\mathcal{A}_{\text{I}}$ outputs a valid forgery.

$\mathcal{A}_{\text{II}}$ makes at most q_H hash queries to oracle H_2 and $\mathcal{B}_{\text{II}}$ randomly selects a $j^* \in [1, q_H]$. Therefore, the identity ID_j in $\mathcal{A}_{\text{II}}$'s forgery satisfies $j = j^*$ with the probability at least $\frac{1}{q_H}$. According to the security model, $\mathcal{A}_{\text{II}}$ does not make the secret value query and public key replace query on the identity ID_j of its forgery. As a result, $\mathcal{B}_{\text{II}}$ does not abort during the simulation with probability at least $\frac{1}{q_H}$. Since $\mathcal{A}_{\text{II}}$ has advantage ϵ in forging a valid signature, $\mathcal{B}_{\text{II}}$ has advantage $\frac{\epsilon}{q_H}$ in solving the CDH problem instance. This completes the proof of (2). □

Theorem 2. *The equivalent validity of the aggregate signature holds against fully chosen-key attacks for the proposed scheme, assuming that the hash function H_4 is collision-resistant.*

Proof of Theorem 2. Suppose there exists an adversary $\mathcal{A}$ who has advantage ϵ in breaking the equivalent validity of the proposed CLAS scheme, then we show there exists a simulator $\mathcal{B}$ that has advantage ϵ in breaking the collision-resistance property of hash function H_4. $\mathcal{B}$ interacts with $\mathcal{A}$ as follows.

Setup. $\mathcal{B}$ runs the master key generation algorithm Setup to generate the master public and secret key pair (*mpk, msk*), where

$$mpk = (\mathbb{PG}, l, str_p, str_q, P, Q, P_{pub}, H_1, H_2, H_3, H_4), \;\; msk = s \in \mathbb{Z}_q.$$

Then, it selects a random $y \in \mathbb{Z}_q$, generates a public and secret verification key pair $(pk_{\mathsf{ver}}, sk_{\mathsf{ver}}) = (yP, y)$, and returns the master public key *mpk* and public verification key pk_{ver} to $\mathcal{A}$.

Private Key Query. On receiving a private key query on identity ID_j that has not been queried before, $\mathcal{B}$ can generate the partial private key d_{ID_j} with the master secret key *msk* and the secret value v_{ID_j} from the user key generation algorithm UserKeyGen. Then, it returns $sk_{\mathsf{ID}_j} = (d_{\mathsf{ID}_j}, v_{\mathsf{ID}_j})$ as the private key of the user with identity ID_j to $\mathcal{A}$.

Aggregation verification query. On receiving an aggregation verification query on an aggregate signature Σ of identities, public keys, and messages $\{(\mathsf{ID}_i, pk_{\mathsf{ID}_i}, m_i) \mid i = 1, \ldots, n\}$, $\mathcal{B}$ runs the aggregation verification algorithm AggVer with the secret verification key $sk_{\mathsf{ver}} = y$ as input to generate the verification result. Then, it returns the verification result to $\mathcal{A}$.

Forgery. At last, $\mathcal{A}$ outputs a set of four-tuples $\{(\mathsf{ID}_i, pk_{\mathsf{ID}_i}, m_i, \sigma_i) \mid i = 1, \ldots, n\}$ consisting of identities, public keys, messages, signatures, and a value Σ supposed to be the aggregate signature on the set of four-tuples.

If $\mathcal{A}$ wins the game, the following conditions are satisfied.

1. The aggregate signature Σ is generated from all the single signatures,

$$\mu = H_4\Big(e(V_1, yP) \parallel \ldots \parallel e(V_n, yP)\Big).$$

2. The aggregate signature Σ is valid, i.e.

$$\mu = H_4\left(\begin{array}{c} e\Big(H_2(\mathsf{ID}_1) + h_1U_1, yP_{pub}\Big) \cdot e(h_1 \cdot pk_{\mathsf{ID}_1} + U_1, yQ) \parallel \ldots \\ \parallel e\Big(H_2(\mathsf{ID}_n) + h_nU_n, yP_{pub}\Big) \cdot e(h_n \cdot pk_{\mathsf{ID}_n} + U_n, yQ) \end{array} \right)$$

3. There exists at least one $i^* \in [1, n]$, such that σ_{i^*} is invalid, i.e.

$$e(V_{i^*}, P) \neq e\Big(H_2(\mathsf{ID}_{i^*}) + h_{i^*}U, P_{pub}\Big) \cdot e(h_{i^*} \cdot pk_{\mathsf{ID}_{i^*}} + U, Q)$$

and hence $e(V_{i^*}, yP) \neq e(H_2(\mathsf{ID}_{i^*}) + h_{i^*}U, yP_{pub}) \cdot e(h_{i^*} \cdot pk_{\mathsf{ID}_{i^*}} + U, yQ)$.

Table 1
Efficiency comparison.

	Aggregation Cost	Aggregation Verification Cost	Aggregate Signature Size
Scheme in [5]	–	$nh + 2ns + 3e$	$(n+1)\mathbb{G}$
Our Scheme	$h + ne$	$nh + 2ns + (2n+3)e$	$(n+1)\mathbb{G} + \mathbb{Z}_q$

Suppose there are n single signatures, h, e, s denote the hash operation, pairing operation, and scalar multiplication in group $\mathbb{G}$, respectively. " – " denotes that the aggregation algorithm of the scheme in [5] only includes group operations, which is omitted in the comparison.

Note that the hash values $H_4(e(V_1, yP) \parallel \ldots \parallel e(V_n, yP))$ and

$$H_4\left(\begin{array}{c} e\Big(H_2(\mathsf{ID}_1) + h_1U_1, yP_{pub}\Big) \cdot e(h_1 \cdot pk_{\mathsf{ID}_1} + U_1, yQ) \parallel \ldots \\ \parallel e\Big(H_2(\mathsf{ID}_n) + h_nU_n, yP_{pub}\Big) \cdot e(h_n \cdot pk_{\mathsf{ID}_n} + U_n, yQ) \end{array}\right)$$

are the same. On the other hand, the two inputs are different since

$$e(V_{i^*}, yP) \neq e\Big(H_2(\mathsf{ID}_{i^*}) + h_{i^*}U, yP_{pub}\Big) \cdot e(h_{i^*} \cdot pk_{\mathsf{ID}_{i^*}} + U, yQ).$$

Therefore, $\mathcal{B}$ presents a pair of collisions of hash function H_4.

This completes the description of the simulation and how $\mathcal{B}$ outputs a pair of collisions. Next, we analyze the advantage of $\mathcal{B}$. It holds the master secret key *msk* and can answer the private key queries. Moreover, $\mathcal{B}$ holds the secret verification key sk_{ver} and hence it can answer the aggregation verification queries. The simulation is indistinguishable from the real scheme. If $\mathcal{A}$ has advantage ϵ in breaking the equivalent validity of the proposed CLAS scheme, $\mathcal{B}$ has advantage ϵ in outputting a pair of collisions of hash function H_4. This completes the proof of 2. □

5. Experimental results

In this section, we give an efficiency comparison of the scheme in [5] and our scheme as listed in Table 1. In addition, we present the experimental results of the computation and communication costs, which include the time cost of generating and verifying an aggregate signature and the size of an aggregate signature with respect to the number of single signatures.

We use the type A curve implemented in Pairing-Based Cryptography library (PBC-0.5.14). The hardware and software specification is as follows.

CPU	Intel Core i5-2500 @3.3 GHz
RAM	8 GB DDR3 1333 MHz
Operation System	64-bit Ubuntu Linux 14.04
Compiler	GNU C&C+ Compiler 4.8.4

In achieving the 80-bit security level, the size of prime p in defining the elliptic curve $E(\mathbb{F}_p)$ is at least 160-bit and the size of p^k in defining the target group $\mathbb{G}_T$ is at least 1024-bit. For type A curves implemented in the PBC library, the embedding degree k equals to 2. Therefore, the size of p is at least 512-bit and the order q of the group is least 160-bit.

p	0xA7A73868E95FBA886EDEF8CE96E7217E364BB946F5 ED839628D1F80010940622A7AFDAF9B049744A459E54D AB7BA5BE92539E8FF9B4F30A3CF6230C28E284D97
Equation	$y^2 = x^3 + x$
q	0x8000000000000800000000000000000000000001

From Figs. 1 and 2, we can see that the aggregation algorithm requires little time cost compared with the aggregation verification algorithm. The aggregation operation of the scheme in [5] is very efficient since only point addition on elliptic curves is required. To reinforce the scheme to withstand the *fully chosen-key attacks*, an additional hash tag is appended to the aggregate signature, computing the hash tag requires pairing operations. The aggregation operation of our scheme takes roughly 1 millisecond per single signature.

As for the aggregation verification algorithm, the efficiency of our scheme is in parallel with the scheme in [5]. The reason is that although more pairing operations are required to compute the hash tag in our scheme, they are very quickly implemented in type A curves. On the other hand, the scalar multiplication operations are comparatively slow and they are required in the aggregation verification algorithm. Therefore, the pairing computation is not the dominating factor that determines the efficiency. In particular, our scheme roughly takes 23% additional time cost to achieve security against *fully chosen-key attacks*.

The communication cost of our scheme is efficient since the size of the aggregate signature is quite compact. From Fig. 3, it easy to see that the size of the aggregate signature is almost half as the sum of all single signatures involved in the aggregation. Compared with the scheme in [5], the aggregate signature in our scheme only contains an additional hash tag,

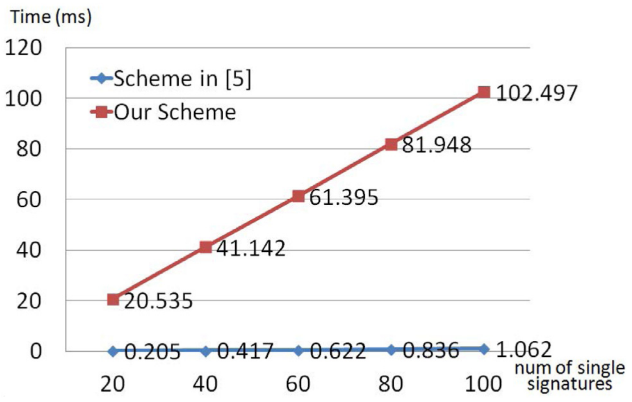

Fig. 1. Time cost of aggregation algorithm.

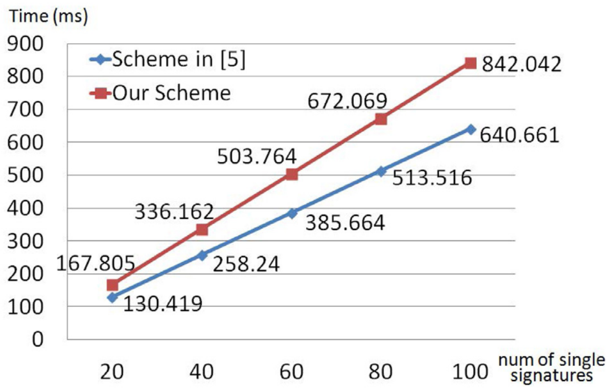

Fig. 2. Time cost of aggregation verification algorithm.

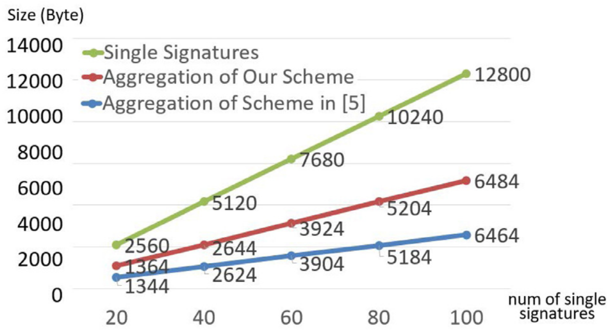

Fig. 3. The size of aggregate signatures.

which is an element from group $\mathbb{Z}_q$. Here, q is the order of the group defined over $E(\mathbb{F}_p)$ which is about 20 Bytes. Therefore, it is almost priceless with respect to the size of the aggregate signature to reinforce the security of a certificateless aggregate signature scheme to withstand the *fully chosen-key attacks*.

6. Conclusion

In this paper, we took an insight into the security of certificateless aggregate signature (CLAS) schemes. We considered a potential attack in the aggregate signature setting which was not considered in the traditional CLAS schemes. This attack was referred to as the *fully chosen-key attacks* and a concrete example is that some dishonest users collude together to cheat the verifier. Therefore, this potential weakness should be taken into consideration in the design of CLAS schemes.

We define the security model against the *fully chosen-key attacks*. A scheme secure in the traditional security model is not necessarily secure against such an attack. Next, we demonstrate how to reinforce the security of the scheme by applying an additional collision-resistant hash function.

In our notion of CLAS schemes, there is an intended verifier who is responsible for verifying the aggregate signatures. The verifier generates the public and secret verification key pair. If a dispute occurs in the future, the verifier can publish the secret verification key so that the controversial aggregate signature can be publicly verified.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work was supported by the National Natural Science Foundation of China under Grant No. 61672289, 61802195, and 61972094. This work was also supported by the Purple Mountain Laboratories for Network and Communication Security.

References

[1] S.S. Al-Riyami, K.G. Paterson, Certificateless public key cryptography, in: C. Laih (Ed.), Advances in Cryptology - ASIACRYPT 2003, Lecture Notes in Computer Science, volume 2894, Springer, 2003, pp. 452–473.

[2] Au M.H., Mu Y., Chen J., Wong D.S., Liu J.K., Yang G., Malicious KGC attacks in certificateless cryptography, ACM, 2007, Proceedings of the ACM Symposium on Information, Computer and Communications Security, 302–311

[3] D. Boneh, C. Gentry, B. Lynn, H. Shacham, Aggregate and verifiably encrypted signatures from bilinear maps, in: E. Biham (Ed.), Advances in Cryptology - EUROCRYPT 2003, Lecture Notes in Computer Science, volume 2656, Springer, 2003, pp. 416–432.

[4] C. Chen, H. Chien, G. Horng, Cryptanalysis of a compact certificateless aggregate signature scheme, I. J. Network Secur. 18 (4) (2016) 793–797.

[5] L. Cheng, Q. Wen, Z. Jin, H. Zhang, L. Zhou, Cryptanalysis and improvement of a certificateless aggregate signature scheme, Inf. Sci. 295 (2015) 337–346.

[6] J. Cui, J. Zhang, H. Zhong, R. Shi, Y. Xu, An efficient certificateless aggregate signature without pairings for vehicular ad hoc networks, Inf. Sci. 451–452 (2018) 1–15.

[7] Fu A., Yu S., Zhang Y., Wang H., Huang C., NPP: A new privacy-aware public auditing scheme for cloud data sharing with group users, IEEE Trans. Big Data (2017), doi: 10.1109/TBDATA.2017.2701347.

[8] Z. Gong, Y. Long, X. Hong, K. Chen, Two certificateless aggregate signatures from bilinear maps, in: W. Feng, F. Gao (Eds.), Proceedings of the International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing - SNPD 2007, IEEE Computer Society, 2007, pp. 188–193.

[9] Z. Gong, Y. Long, X. Hong, K. Chen, Practical certificateless aggregate signatures from bilinear maps, J. Inf. Sci. Eng. 26 (6) (2010) 2093–2106.

[10] D. He, M. Tian, J. Chen, Insecurity of an efficient certificateless aggregate signature with constant pairing computations, Inf. Sci. 268 (2014) 458–462.

[11] S. Horng, S. Tzeng, P. Huang, X. Wang, T. Li, M.K. Khan, An efficient certificateless aggregate signature with conditional privacy-preserving for vehicular sensor networks, Inf. Sci. 317 (2015) 48–66.

[12] X. Huang, Y. Mu, W. Susilo, D.S. Wong, W. Wu, Certificateless signature revisited, in: J. Pieprzyk, H. Ghodosi, E. Dawson (Eds.), Proceedings of the Australasian Conference on Information Security and Privacy - ACISP 2007, Lecture Notes in Computer Science, volume 4586, Springer, 2007, pp. 308–322.

[13] J. Kim, H. Oh, FAS: Forward secure sequential aggregate signatures for secure logging, Inf. Sci. 471 (2019) 115–131.

[14] J. Li, H. Yuan, Y. Zhang, Cryptanalysis and improvement for certificateless aggregate signature, Fundam. Inform. 157 (1–2) (2018) 111–123.

[15] Y. Liu, X. Hu, W. Tan, Study on a provably secure certificateless aggregate signature scheme, in: Proceedings of the International Conference on Networking, Sensing, and Control - ICNSC 2016, IEEE, 2016, pp. 1–4.

[16] A.K. Malhi, S. Batra, An efficient certificateless aggregate signature scheme for vehicular ad-hoc networks, Discrete Math. Theoret. Comput. Sci. 17 (1) (2015) 317–338.

[17] D. Pointcheval, J. Stern, Security proofs for signature schemes, in: U.M. Maurer (Ed.), Advances in Cryptology - EUROCRYPT 1996, Lecture Notes in Computer Science, volume 1070, Springer, 1996, pp. 387–398.

[18] Y. Qu, Q. Mu, An efficient certificateless aggregate signature without pairing, Int. J. Electron. Secur. Digital Forens. 10 (2) (2018) 188–203.

[19] H. Shen, J. Chen, H. Hu, J. Shen, Insecurity of a certificateless aggregate signature scheme, IEICE Trans. 99-A (2) (2016) 660–662.

[20] H. Shen, J. Chen, J. Shen, D. He, Cryptanalysis of a certificateless aggregate signature scheme with efficient verification, Secur. Commun. Netw. 9 (13) (2016) 2217–2221.

[21] K. Shim, Security models for certificateless signature schemes revisited, Inf. Sci. 296 (2015) 315–321.

[22] L. Wang, K. Chen, Y. Long, H. Wang, Cryptanalysis of a certificateless aggregate signature scheme, Secur. Commun. Netw. 9 (11) (2016) 1353–1358.

[23] L. Wu, Z. Xu, D. He, X. Wang, New certificateless aggregate signature scheme for healthcare multimedia social network on cloud environment, Security and Communication Networks 2018 (2018) 2595273:1–2595273:13.

[24] H. Xiong, Z. Guan, Z. Chen, F. Li, An efficient certificateless aggregate signature with constant pairing computations, Inf. Sci. 219 (2013) 225–235.

[25] F. Zhang, L. Shen, G. Wu, Notes on the security of certificateless aggregate signature schemes, Inf. Sci. 287 (2014) 32–37.

[26] H. Zhang, Insecurity of a certificateless aggregate signature scheme, Secur. Commun. Netw. 9 (11) (2016) 1547–1552.

[27] J. Zhang, X. Zhao, J. Mao, Attack on chen *et al.*'s certificateless aggregate signature scheme, Secur. Commun. Netw. 9 (1) (2016) 54–59.

[28] L. Zhang, F. Zhang, Security model for certificateless aggregate signature schemes, in: Proceedings of the International Conference on Computational Intelligence and Security - CIS 2008, IEEE Computer Society, 2008, pp. 364–368.