Certificateless Public Auditing for Data Integrity in the Cloud

Boyang Wang[†,‡], Baochun Li[‡], Hui Li[†] and Fenghua Li[†,§]
[†] State Key Laboratory of Integrated Service Networks, Xidian University, Xi'an, Shaanxi, China
[‡] Department of Electrical and Computer Engineering, University of Toronto, Toronto, Ontario, Canada
[§] State Key Laboratory of Information Security, Chinese Academy of Sciences, Beijing, China

Abstract—**Due to the existence of security threats in the cloud, many mechanisms have been proposed to allow a user to audit data integrity with the public key of the data owner before utilizing cloud data. The correctness of choosing the right public key in previous mechanisms depends on the security of Public Key Infrastructure (PKI). Although traditional PKI has been widely used in the construction of public key cryptography, it still faces many security risks, especially in the aspect of managing certificates. In this paper, we design a certificateless public auditing mechanism to eliminate the security risks introduced by PKI in previous solutions. Specifically, with our mechanism, a public verifier does not need to manage certificates to choose the right public key for the auditing. Instead, the auditing can be operated with the assistance of the data owner's identity, such as her name or email address, which can ensure the right public key is used. Meanwhile, this public verifier is still able to audit data integrity without retrieving the entire data from the cloud as previous solutions. To the best of our knowledge, it is the first certificateless public auditing mechanism for verifying data integrity in the cloud. Our theoretical analyses prove that our mechanism is correct and secure, and our experimental results show that our mechanism is able to audit the integrity of data in the cloud efficiently.**

I. Introduction

Nowadays, as the cloud offers data storage services with much lower prices than the cost of maintaining data on personal devices, people tend to outsource the hosting of their data to the cloud. By enjoying such storage services in the cloud, data owners are able to freely access their outsourced data on different devices and locations, and easily share their data with others. Although cloud providers have designed a series of security protections for these data storage services, casting the image of a more reliable and secure place to store data than personal devices, the integrity of data stored in the cloud may still be in doubt due to the existence of hardware/software failures and human errors [1], [2]. For example, Dropbox, a well-known cloud-based data storage service with over 100 million users, accidentally allowed anybody to access Dropbox accounts without passwords for several hours after an unsuccessful code update in June 2011 [3].

To efficiently audit data integrity in an untrusted cloud, many mechanisms have been proposed [2], [4]–[16]. One of the most attractive features of these works is allowing not only the data owner herself but also a public verifier, such as a data user who would like to utilize cloud data, to verify the integrity of cloud data without retrieving the entire data from the cloud, referred to as *public auditing*. Another common feature of these previous works is that choosing the correct public key of the data owner during the verification on cloud data integrity is based on the security of Public Key Infrastructure (PKI).

In traditional PKI, the assurance of the binding between an owner's identity and her public/private key is delivered by the Certificate Authority (CA) and certificates issued by the CA. Although PKI has been widely used in the construction of public key cryptography, it still faces many security risks [17]–[19]. One of the most fundamental issues is the management of certificates, including distribution, storage, revocation and verification. For example, a certificate can only be trusted by users if the root certificate of this certificate is trustworthy; however, since the root certificate is self-signed by a CA itself, to determine the trustworthiness of this root certificate in the first place is not an easy task, even for a security expert [17]. It is an even harder — and sometimes confusing — process for the general public, who have no special knowledge of cryptography and security. All they can do is perhaps to click the button shows Accept, and install a so-called "trustworthy" certificate anyway.

Considering these security risks, the certificate of a data owner that a public verifier (i.e., a data user) obtains may not be trustworthy, and the public key used for verifying cloud data integrity may not even belong to the expected data owner. In this case, even the verification result is positive, the cloud data that a public verifier intend to utilize may not be actually signed by the data owner herself. Note that some symmetric key-based solutions [20], [21] can certainly be leveraged to verify the correctness of data stored in an untrusted cloud without involving certificates. However, they are not public verifiable. Therefore, how to avoid managing certificates at public verifiers while still designing a public key-based mechanism to securely and efficiently audit data integrity in the cloud is a necessary task.

To avoid managing certificates in a public auditing mechanism, utilizing Identity-Based Signatures (IBS) [22], [23] seems to be an option in the first place. Unfortunately, IBS has an inherent drawback — *the key escrow problem* [19]. By leveraging the existing technique of certificateless signatures (CLS) [19], a public verifier should be able to audit data integrity without managing certificates or suffering the key escrow problem. In particular, a public verifier should be able to leverage the owner's identity, such as her name or email address, to ensure the right public key of this owner is used during the auditing of cloud data integrity. However, the main challenge of building a certificateless public auditing mechanism in the cloud is that, traditional certificateless signature schemes [19], [24]–[26] cannot satisfy one of the most significant features that a public auditing mechanism should be capable of — verifying the integrity without downloading the entire data, which is referred to as *blockless verifiability*.

In this paper, we first design a homomorphic authenticable certificateless signature scheme with blockless verifiability, which traditional certificateless signature schemes do not support. We then build the entire certificateless public auditing mechanism for verifying data integrity in an untrusted cloud based on our proposed certificateless signature scheme. As a result, our public auditing mechanism does not require a public verifier to manage certificates, which successfully eliminates the security risks introduced by PKI in previous works. Meanwhile, this public verifier is still able to efficiently audit the correctness of data in the cloud without retrieving the entire data. To the best of our knowledge, our mechanism represents the first solution of certificateless public auditing on data integrity in the cloud.

The remainder of this paper is organized as follows. In Sec. II, we present the system and threat model. In Sec. III, we briefly introduce cryptographic primitives used in our mechanism. The detailed design and security analysis of our mechanism are presented in Sec. IV and Sec. V. Sec. VI evaluates the performance of our mechanism. Finally, we discuss related work in Sec. VII, and conclude this paper in Sec. VIII.

II. Problem Statement

As presented in Fig. 1, the system model in this paper includes four entities: the cloud, the data owner, data users and the Key Generation Center (KGC). The cloud provides data services to the data owner and data users. The data owner outsources her data to the cloud and save her storage on local devices. In general, in order to be modified efficiently, the outsourced data is further divided into a number of blocks. A data user is able to utilize cloud data outsourced by the data owner via the services in the cloud. For instance, a data user can perform search or computation on cloud data for particular purposes. The KGC is a trusted party required in the framework of certificateless schemes [19], [24]–[26]. It is able to generate a partial private key of an entity (e.g., the data owner) based on the corresponding identity (e.g., name or email address). The remaining part of the entire private key is generated by the entity itself.

The data stored in the cloud may be polluted from two possible causes. First, an external adversary may try to pollute data, and prevent the owner and users from using the data correctly. Second, cloud service providers may accidentally corrupt data integrity due to hardware/software failures or human errors, and lie about data corruption to save the reputation of their services. As a result, the data owner and data users do not fully trust the cloud with the integrity of data.

To protect data integrity, each block is attached with a signature, which is computed by the owner's entire private key. A data user needs to check cloud data integrity before any utilization (e.g., search, computation, data mining). Specifically, a data user first sends an auditing challenge to the cloud. Then, the cloud generates a proof of possession of the owner's data as an auditing response to this data user. Finally, this data user verifies data integrity based on the auditing response with the public key of the data owner and the owner's identity. Essentially, the process of public auditing is a challenge-and-response protocol between a data user and the cloud. Note that the data owner herself can also be a verifier to check the integrity of data, which she on longer physically possesses, by following the same protocol.

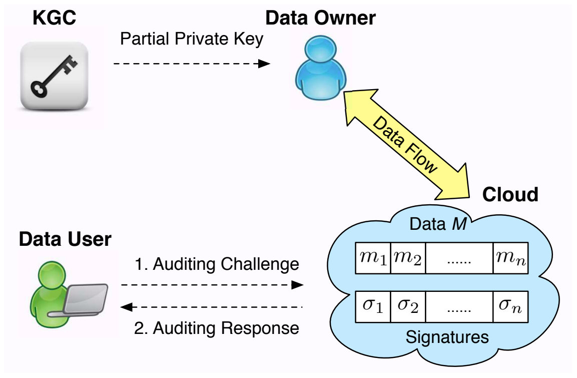


Fig. 1. The cloud, the data owner, data users and the KGC.

The design of our public auditing mechanism should achieve three objectives: (1) **Correctness:** A public verifier (i.e., a data user) is able to verify the integrity of data in the cloud correctly. (2) **Public Auditing:** A public verifier is able to audit the correctness of data without retrieving the entire data from the cloud. (3) **Certificateless:** The correctness of public auditing does not require a public verifier to manager certificates.

III. Preliminaries

A. Bilinear Maps

Let $\mathbb{G}_1$ and $\mathbb{G}_2$ be two multiplicative cyclic groups of prime order p, P be a generator of $\mathbb{G}_1$. Bilinear map e is a map $\mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_2$ with the following properties: (1) **Computability**: there exists an efficient algorithm for computing map e. (2) **Bilinearity**: $e(U^a, V^b) = e(U, V)^{ab}$, for all $U, V \in \mathbb{G}_1$, and $a, b \in \mathbb{Z}_p$. (3) **Non-degeneracy**: $e(P, P) \neq 1$.

B. Complexity Assumptions

Definition 1: **Computational Diffie-Hellman (CDH) Assumption.** Let $a, b \in \mathbb{Z}_p^*$, given $P, P^a, P^b \in \mathbb{G}_1$ as input, for any *probabilistic polynomial time* adversary $\mathcal{A}_{\mathcal{CDH}}$, it is *computational infeasible* to output P^{ab}, which is defined as

$$Pr[\mathcal{A}_{\mathcal{CDH}}(P, P^a, P^b) = < P^{ab} >: a, b \overset{R}{\leftarrow} \mathbb{Z}_p^*] \leq \epsilon,$$

where ϵ is *negligible*.

Definition 2: **Discrete Logarithm (DL) Assumption.** Let $a \in \mathbb{Z}_p^*$, given $P, P^a \in \mathbb{G}_1$ as input, for any *probabilistic polynomial time* adversary $\mathcal{A}_{\mathcal{DL}}$, it is *computational infeasible* for it to output a, which is defined as

$$Pr[\mathcal{A}_{\mathcal{DL}}(P, P^a) = < a >: a \overset{R}{\leftarrow} \mathbb{Z}_p^*] \leq \epsilon,$$

where ϵ is *negligible*.

C. Homomorphic Authenticable Signatures

Homomorphic authenticable signatures, also referred to as homomorphic verifiable tags or homomorphic authenticators, are fundamental building blocks in the construction of public auditing mechanisms [2], [4]–[15]. One of the most important

Setup. Given security parameter δ, the KGC outputs $(P, \mathbb{G}_1, \mathbb{G}_2, e)$, where P is a generator of $\mathbb{G}_1$, $\mathbb{G}_1$ and $\mathbb{G}_2$ are two cyclic multiplicative groups of prime order p and $e: \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_2$ is a bilinear map.

The KGC chooses a random $\lambda \in \mathbb{Z}_p^*$ as the *master key* and sets $P_T = P^{\lambda}$. The KGC also chooses a random $P_1 \in \mathbb{G}_1$, and two cryptographic hash functions $H_1 : \{0,1\}^* \rightarrow \mathbb{G}_1$ and $H_2 : \{0,1\}^* \rightarrow \mathbb{G}_1$. The system parameters are ($\mathbb{G}_1$, $\mathbb{G}_2$, e, P, P_T, P_1, H_1, H_2), which are public. The KGC keeps the master key private.

Partial-Private-Key-Extract. Given signer $\mathcal{S}$'s identifier $ID_s \in \{0,1\}^*$, the KGC generates the *partial private key* for signer $\mathcal{S}$ with its master key λ:

1) Compute $Q_s = H_1(ID_s) \in \mathbb{G}_1$.
2) Output the partial private key $D_s = Q_s^{\lambda} \in \mathbb{G}_1$.

KeyGen. Given system parameters ($\mathbb{G}_1$, $\mathbb{G}_2$, e, P, P_T, P_1, H_1, H_2), signer $\mathcal{S}$ chooses a random $x_s \in \mathbb{Z}_p^*$ as her *secret key*, and also computes $P_s = P^{x_s} \in \mathbb{G}_1$ as her *public key*. The entire private key of signer $\mathcal{S}$ includes partial private key D_s and secret key x_s.

Sign. Given block $m \in \mathbb{Z}_p$ and block identifier $id \in \{0,1\}^*$, signer $\mathcal{S}$ computes a signature using partial private key D_s and secret key x_s as follows:

1) Compute $V = H_2(ID_s||P_s||id) \cdot P_1^m \in \mathbb{G}_1$.
2) Output a signature σ on block m and block identifier id as $\sigma = V^{x_s} \cdot D_s \in \mathbb{G}_1$.

Verify. Given system parameters ($\mathbb{G}_1$, $\mathbb{G}_2$, e, P, P_T, P_1, H_1, H_2), public key P_s, signer identifier ID_s, block m, block identifier id and signature σ, a verifier checks the integrity of this block as:

1) Compute $Q_s = H_1(ID_s)$.
2) Compute $V = H_2(ID_s||P_s||id) \cdot P_1^m$.
3) Verify the following equation

$$e(\sigma, P) \overset{?}{=} e(Q_s, P_T) \cdot e(V, P_s). \quad (1)$$

If the equation holds, output *valid*. Otherwise, output *invalid*.

Fig. 2. Details of HA-CLS.

features of homomorphic authenticable signatures is blockless verifiability.

With blockless verifiability, a verifier is able to check the integrity of data stored in the cloud by retrieving a single block (which is a combination of all the blocks in data) instead of downloading the entire data. Because the size of data in the cloud is normally very large, this advanced property saves a verifier amount of bandwidth and offers it an efficient and secure solution of verifying the correctness of cloud data outsourced by the data owner. Another important properties of homomorphic authenticable signatures is *non-malleable* [14]. Non-malleable indicates that an untrusted cloud is not able to generate valid signatures on combined blocks by combining existing signatures.

D. Certificateless Signatures

Certificateless signatures (CLS), first proposed by Al-Riyami and Paterson [19], are able to avoid asking entities to manage certificates in the construction of public key cryptography. In addition, certificateless signatures do not have the key escrow problem, which is an inherent drawback in Identity-Based Signatures (IBS) [23].

More specifically, in IBS, the entire private key of an entity is independently generated by the KGC, then the KGC has the ability of computing any entity's signatures, which is referred to as the key escrow problem. While in certificateless signature schemes, the KGC is only responsible for generating a partial private key to an entity, and the remaining part of the entire private key is generated by the entity itself. Therefore, the KGC in certificateless signature schemes cannot compute a signature of any entity, because it does not have the knowledge of the entire private key.

IV. Homomorphic Authenticable CLS

A. Overview

As we mentioned in the introduction, the key idea of this paper is to avoid asking verifiers to manage certificates in the design of a public auditing mechanism by leveraging certificateless signatures. Unfortunately, an important challenge of designing the entire public auditing mechanism without managing certificates is that traditional certificateless signature schemes [19], [24]–[26] are not blockless verifiable. That means if we directly apply these traditional certificateless signature schemes to the public auditing mechanism, a verifier has to download the entire data from the cloud to check the integrity, which is not efficient.

Therefore, we first propose a novel homomorphic authenticable certificateless signature scheme (named HA-CLS), which is blockless verifiable and non-malleable. Then, based on the design of this proposed certificateless signature scheme, we will build the entire certificateless public auditing mechanism for cloud users in the next section.

B. Design of HA-CLS

Our proposed homomorphic authenticable certificateless signature scheme (HA-CLS) includes five algorithms: **Setup**, **Partial-Private-Key-Extract**, **KeyGen**, **Sign** and **Verify**.

In **Setup**, the KGC generates a *master key* and system parameters. The KGC is able to generate *partial private keys* for signers in **Partial-Private-Key-Extract**. In **KeyGen**, a signer is able to compute a *secret key* and a *public key* for herself. In **Sign**, a signer is able to compute signatures on blocks with her *entire private key*, which includes her partial private key and secret key. In **Verify**, a verifier can check the correctness of a signature by using the public key of the signer and the identity of this signer. Details of each algorithm are presented as Fig. 2.

The correctness of Equation 1 in **Verify** can be proved by using the properties of bilinear maps. More specifically, we have

$$\begin{aligned} e(Q_s, P_T) \cdot e(V, P_s) &= e(Q_s, P^{\lambda}) \cdot e(V, P^{x_s}) \\ &= e(Q_s^{\lambda}, P) \cdot e(V^{x_s}, P) \\ &= e(D_s \cdot V^{x_s}, P) \\ &= e(\sigma, P). \end{aligned}$$

Note that we have both signer identifiers and block identifiers in the design of our certificateless signature scheme. Generally, a signer identifier is the name or email address of this signer, and a block identifier is able to distinguish this block from other blocks in the entire data. To distinguish these two types of identifiers from each other, in this paper, signer identifiers are all described with uppercase (e.g. ID) and block identifiers are all presented with lowercase (e.g. id).

C. Security Analysis of HA-CLS

We now discuss the security properties of our homomorphic authenticable certificateless signature scheme, including unforgeability, blockless verifiability, and non-malleability.

***Theorem** 1: It is computationally infeasible to generate a forgery of a signature with HA-CLS.*

Proof: As defined in [19], [24], two types of adversaries should be considered with the standard security model of a certificateless signature scheme. These two types of adversaries, denoted as Type-I Adversary and Type-II Adversary respectively, have different attack capabilities. Detailed definitions of these two types of adversaries are presented as follows:

- **Type-I Adversary:** This type of adversaries $\mathcal{A}_I$ does not have access to the master key of the KGC, but $\mathcal{A}_I$ has the ability to replace the public key of any entity with a value of its choice (the reason that $\mathcal{A}_I$ has this ability is because there is no certificates involved in the certificateless signature scheme).
- **Type-II Adversary:** This type of adversaries $\mathcal{A}_{II}$ has access to the master key of the KGC, but it cannot replace the public key of any entity (the success of $\mathcal{A}_{II}$ will indicate the existence of the key escrow problem in the certificateless signature scheme).

We will prove that if Type-I Adversary $\mathcal{A}_I$ or Type-II Adversary $\mathcal{A}_{II}$ is able to generate a forgery of a signature with HA-CLS, then there exists an algorithm $\mathcal{F}$ that is able to solve the CDH problem in $\mathbb{G}_1$ (given P, P^a and P^b, output P^{ab}), which will contradict to the assumption that the CDH problem is computationally infeasible in $\mathbb{G}_1$. Let us first consider about the case of Type-I Adversary.

Type-I Adversary: Based on the construction of HA-CLS, to generate a forgery of a signature in a security game simulated by algorithm $\mathcal{F}$, $\mathcal{A}_I$ needs to request five different types of queries to algorithm $\mathcal{F}$, including setup query, hash-I query, partial-private-key-extract query, hash-II query, and signing query. Meanwhile, $\mathcal{A}_I$ is able to perform public key replacement in the game. In this game, hash-I (i.e. $H_1 : \{0,1\}^* \rightarrow \mathbb{G}_1$) is treated as a random oracle. Given P, P^a and P^b, algorithm $\mathcal{F}$ simulates the game as follows:

Setup Query: $\mathcal{A}_I$ requests the setup of the system. $\mathcal{F}$ sets $P_T = P^a$, outputs and returns the entire system parameters ($\mathbb{G}_1$, $\mathbb{G}_2$, e, P, P_T, P_1, H_1, H_2) to $\mathcal{A}_I$.

Hash-I Query: $\mathcal{A}_I$ requests the result of the hash-I query on signer identifier ID_s. $\mathcal{F}$ chooses a random $r \in \mathbb{Z}_p$, and tosses a coin. The coin shows 1 with a probability of p_c and 0 otherwise. If the result of the coin toss is 1, $\mathcal{F}$ sets $H_1(ID_s) = P^r \in \mathbb{G}_1$; if the result of the coin toss is 0, $\mathcal{F}$ sets $H_1(ID_s) = (P^b)^r \in \mathbb{G}_1$. Finally, $\mathcal{F}$ returns the result of $H_1(ID_s)$ to $\mathcal{A}_I$.

Since $\mathbb{G}_1$ is a cyclic group, r is a random element of $\mathbb{Z}_p$, P and P^b are both elements of $\mathbb{G}_1$, P^r and $(P^b)^r$ have the identical distribution in $\mathbb{G}_1$, then $\mathcal{A}_I$ cannot distinguish the result of the coin toss based on the result of $H_1(ID_s)$ returned by $\mathcal{F}$.

Partial-Private-Key-Extract Query: $\mathcal{A}_I$ requests a partial private key on signer identifier ID_s. If the result of the corresponding coin toss in the previous hash-I query was 1, $\mathcal{F}$ outputs the partial private key as $D_s = (P^a)^r$ because $D_s = H_1(ID_s)^a = (P^r)^a = (P^a)^r$, where r was randomly picked in the corresponding hash-I query. Otherwise, $\mathcal{F}$ outputs $\perp$.

Public Key Replacement: According to the assumption of Type-I Adversary, $\mathcal{A}_I$ is able to replace the public key of any entity. More specifically, $\mathcal{A}_I$ first generates a random $x_s \in Z_p^*$, and sets the public key of signer $\mathcal{S}$ as $P_s = P^{x_s}$. Then, $\mathcal{A}_I$ submits (ID_s, x_s, P_s) to $\mathcal{F}$. $\mathcal{F}$ will record this key replacement, which will be used later.

Hash-II Query: $\mathcal{A}_I$ requests the result of the hash-II query on signer identifier ID_s, this signer's public key P_s, block m and block identifier id. $\mathcal{F}$ outputs $V = H_2(ID_s||P_s||id) \cdot P_1^m$, and returns the result of V to $\mathcal{A}_I$.

Signing Query: $\mathcal{A}_I$ requests a signature of signer $\mathcal{S}$ on block m and block identifier id by submitting the result of V, which was returned from the previous hash-II query. If the result of the corresponding coin toss in the previous hash-I query was 1, then $\mathcal{F}$ outputs the signature as $\sigma = V^{x_s}(P^a)^r$, where r was randomly picked in the corresponding hash-I query. Otherwise, $\mathcal{F}$ outputs $\perp$.

Eventually, $\mathcal{A}_I$ outputs a forgery σ on (ID_s, m, id). Then, $\mathcal{F}$ learns that the result of the corresponding hash-I query of this forgery was $H_1(ID_s) = (P^b)^r$, and the forgery is $\sigma = V^{x_s}(P^{ab})^r$. Clearly, $\mathcal{F}$ can output P^{ab} by computing

$$P^{ab} = (\sigma / V^{x_s})^{r^{-1}},$$

because $\mathcal{F}$ knows the values of (σ, V^{x_s}, r) based on the results of queries in the game. It means if $\mathcal{A}_I$ successfully generates a forgery of a signature, then $\mathcal{F}$ is able to solve the CDH problem in $\mathbb{G}_1$ (given P, P^a and P^b, output P^{ab}).

Type-II Adversary: Now, let us consider about the case of Type-II Adversary $\mathcal{A}_{II}$. To generate a forgery of a signature in a security game simulated by algorithm $\mathcal{F}$, $\mathcal{A}_{II}$ also needs to request five different types of queries, including setup query, hash-I query, partial-private-key-extract query, hash-II query, and signing query. Different from the game with a Type-I Adversary, $\mathcal{F}$ should return the master key to $\mathcal{A}_{II}$, however, $\mathcal{A}_{II}$ cannot perform public key replacement. In this game, hash-II (i.e. $H_2 : \{0,1\}^* \rightarrow \mathbb{G}_1$) is treated as a random oracle. Given P, P^a and P^b, algorithm $\mathcal{F}$ simulates the game as follows:

Setup Query: $\mathcal{A}_{II}$ requests the setup of the system. $\mathcal{F}$ generates a random $\lambda \in Z_p^*$ as the master-key and system parameters ($\mathbb{G}_1$, $\mathbb{G}_2$, e, P, P_T, P_1, H_1, H_2). Then, $\mathcal{F}$ returns the master key and system parameters to $\mathcal{A}_{II}$.

Hash-I Query: $\mathcal{A}_{II}$ requests the result of the hash-I query on signer identifier ID_s. $\mathcal{F}$ computes $Q_s = H_1(ID_s) \in \mathbb{Z}_p$ and returns the result of Q_s to $\mathcal{A}_{II}$.

Partial-Private-Key-Extract Query: $\mathcal{A}_{II}$ requests the partial private key on signer identifier ID_s. $\mathcal{F}$ computes the partial private key as $D_s = Q_s^{\lambda}$ and returns it to $\mathcal{A}_{II}$.

As the definition of Type-II Adversary, $\mathcal{A}_{II}$ cannot perform public key replacement. $\mathcal{F}$ sets P^a as the public key of signer $\mathcal{S}$.

Hash-II Query: $\mathcal{A}_{II}$ requests the result of the hash-II query on signer identifier ID_s, this signer's public key P_s, block m and block identifier id. $\mathcal{F}$ generates a random $r \in \mathbb{Z}_p$, and tosses a coin. The coin shows 1 with a probability of p_c and 0 otherwise. If the result of the coin toss shows 1, $\mathcal{F}$ sets $H_2(ID_s||P_s||id) \cdot P_1^m = P^r$; if the result of the coin toss is 0, $\mathcal{F}$ sets $H_2(ID_s||P_s||id) \cdot P_1^m = (P^b)^r$. Finally, $\mathcal{F}$ returns the result of $H_2(ID_s||P_s||id) \cdot P_1^m$ to $\mathcal{A}_{II}$.

Since $\mathbb{G}_1$ is a cyclic group, r is a random element of $\mathbb{Z}_p$, P and P^b are both elements of $\mathbb{G}_1$, P^r and $(P^b)^r$ have the identical distribution in $\mathbb{G}_1$, then $\mathcal{A}_{II}$ cannot distinguish the result of the coin toss based on the result of the hash-II query returned by $\mathcal{F}$.

Signing Query: $\mathcal{A}_{II}$ requests a signature of signer $\mathcal{S}$ on block m and block identifier id. If the result of the corresponding coin toss in the previous hash-II query was 1, then $\mathcal{F}$ outputs the signature as $\sigma = (P^a)^r D_s$, because $\sigma = [H_2(ID_s||P_s||id) \cdot P_1^m]^a D_s = (P^r)^a D_s = (P^a)^r D_s$, where r was randomly picked in the corresponding hash-II query. Otherwise, $\mathcal{F}$ outputs $\perp$.

Eventually, $\mathcal{A}_{II}$ outputs a forgery σ on (ID_s, m, id). Then, $\mathcal{F}$ learns that the result of the corresponding hash-II query of this forgery was $H_2(ID_s||P_s||id) \cdot P_1^m = (P^b)^r$, and the forgery is $\sigma = (P^{ab})^r D_s$. Clearly, $\mathcal{F}$ can output P^{ab} by computing

$$P^{ab} = (\sigma/D_s)^{r^{-1}},$$

because $\mathcal{F}$ knows the values of (σ, D_s, r) based on the records of queries in the game. It means if $\mathcal{A}_{II}$ successfully generates a forgery of a signature, then $\mathcal{F}$ is able to solve the CDH problem in $\mathbb{G}_1$ (given P, P^a and P^b, output P^{ab}).

As discussed above, if $\mathcal{A}_I$ or $\mathcal{A}_{II}$ is able to successfully generate a forgery of a signature, then $\mathcal{F}$ is able to solve the CDH problem in $\mathbb{G}_1$, which contradicts to the assumption that the CDH problem is computationally infeasible in $\mathbb{G}_1$. Therefore, it is computationally infeasible to generate a forgery of a signature with HA-CLS. ■

Theorem 2: *HA-CLS is a homomorphic authenticable certificateless signature scheme.*

Proof: According to the properties we introduced in Section III, to prove HA-CLS is homomorphic authenticable, we need to show that it is not only blockless verifiable but also non-malleable.

To prove the blockless verifiability of HA-CLS, we need to show that a verifier can check the integrity of n blocks by checking the correctness of one combined block. Specifically, given n block identifier $(id_1, ..., id_n)$, n corresponding signatures $(\sigma_1, ..., \sigma_n)$ signed by ID_s, and n random numbers $(y_1, ..., y_n)$, where $y_i \in \mathbb{Z}_p$, a verifier is able to check the correctness of a combined block m', where $m' = \sum_{i=1}^n y_i m_i$, by verifying:

$$e(\prod_{i=1}^n \sigma_i^{y_i}, P) \stackrel{?}{=} e(\prod_{i=1}^n Q_s^{y_i}, P_T) \cdot e(\prod_{i=1}^n W_i^{y_i} \cdot P_1^{m'}, P_s), \quad (3)$$

where $Q_s = H_1(ID_s)$ and $W_i = H_2(ID_s||P_s||id_i)$. Based on the properties of bilinear maps and the correctness of Equation 1, the correctness of Equation 3 can be proved as follows:

$$\begin{aligned} & e(\prod_{i=1}^n Q_s^{y_i}, P_T) \cdot e(\prod_{i=1}^n W_i^{y_i} \cdot P_1^{m'}, P_s) \\ = \ & e(\prod_{i=1}^n Q_s^{y_i}, P^\lambda) \cdot e(\prod_{i=1}^n W_i^{y_i} \cdot P_1^{\sum_{i=1}^n y_i m_i}, P^{x_s}) \\ = \ & e(\prod_{i=1}^n (Q_s^\lambda)^{y_i}, P) \cdot e(\prod_{i=1}^n W_i^{y_i} \cdot \prod_{i=1}^n (P_1^{m_i})^{y_i}, P^{x_s}) \\ = \ & e(\prod_{i=1}^n D_s^{y_i}, P) \cdot e(\prod_{i=1}^n (V_i^{x_s})^{y_i}, P) \\ = \ & e(\prod_{i=1}^n (V_i^{x_s} D_s)^{y_i}, P) = e(\prod_{i=1}^n \sigma_i^{y_i}, P). \end{aligned}$$

The correctness of this combined block m' is based on the correctness of all the n blocks $(m_1, ..., m_n)$. Therefore, we are able to check the integrity of n blocks by verifying the integrity of one combined block, which indicates that HA-CLS is blockless verifiable.

Meanwhile, we can also prove that an adversary, who does not have a private key, cannot generate a valid signature σ' on the combined block m' by combining existing signatures, which indicates the non-malleability of HA-CLS. The hardness of this problem lies in the fact that the cryptographic hash function H_2 must be a one-way function (it is easy to compute every input; however, given the image of a random input, it is hard to invert). More specifically, for the hash function H_2, given a hash value $h \in \mathbb{G}_1$, it should be difficult to find any string $\Omega \in \{0,1\}^*$ such that $h = H_2(\Omega)$.

To prove HA-CLS is non-malleable, we first assume the adversary is able to successfully generate a valid signature by combining existing signatures. More concretely, given two pairs of block and block identifier (m_1, id_1) and (m_2, id_2), two corresponding signatures σ_1 and σ_2, a combined block $m' = m_2 + m_2$, this combined block's identifier id' and signature σ', then according to our assumption, we have

$$\begin{cases} \sigma' = \sigma_1 \sigma_2 \\ \sigma' = V'^{x_s} D_s \\ \sigma_1 \sigma_2 = (V_1 V_2)^{x_s} D_s^2. \end{cases}$$

Based on the above equations, we can further have

$$(\frac{H_2(ID_s||P_s||id')}{H_2(ID_s||P_s||id_1) \cdot H_2(ID_s||P_s||id_2)})^{x_s} = D_s.$$

Similarly, for another pair of blocks m_3 and m_4, and another combined block $m'' = m_3 + m_4$, we can also have

$$\begin{cases} \sigma'' = \sigma_3 \sigma_4 \\ \sigma'' = V''^{x_s} D_s \\ \sigma_3 \sigma_4 = (V_3 V_4)^{x_s} D_s^2, \end{cases}$$

and

$$(\frac{H_2(ID_s||P_s||id'')}{H_2(ID_s||P_s||id_3) \cdot H_2(ID_s||P_s||id_4)})^{x_s} = D_s.$$

Finally, we can learn that

$$H_2(ID_s||P_s||id') = \frac{H_2(ID_s||P_s||id'') \cdot W_1 \cdot W_2}{W_3 \cdot W_4},$$

Setup. Given security parameter δ, the KGC outputs $(P, \mathbb{G}_1, \mathbb{G}_2, e)$, where P is a generator of $\mathbb{G}_1$, $\mathbb{G}_1$ and $\mathbb{G}_2$ are two cyclic multiplicative groups of prime order p and $e : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_2$ is a bilinear map. The KGC chooses a random $\lambda \in \mathbb{Z}_p^*$ as the master key and sets $P_T = P^{\lambda}$.

The KGC also chooses k random elements $(P_1, ..., P_k) \in \mathbb{G}_1^k$ as the *public aggregated key*, and two cryptographic hash functions $H_1 : \{0,1\}^* \rightarrow \mathbb{G}_1$ and $H_2 : \{0,1\}^* \rightarrow \mathbb{G}_1$. The system parameters are ($\mathbb{G}_1$, $\mathbb{G}_2$, e, P, P_T, P_1, ..., P_k, H_1, H_2), which are public. The KGC keeps the master key private.

Data owner $\mathcal{O}$'s data M, which will be stored in the cloud, is divided into n blocks, and each block contains k elements of $\mathbb{Z}_p$. Then, data M can be presented as $M = (\boldsymbol{m}_1, ..., \boldsymbol{m}_n)$, where $\boldsymbol{m}_i = (m_{i,1}, ..., m_{i,k}) \in \mathbb{Z}_p^k$.

Partial-Private-Key-Extract. Given data owner $\mathcal{O}$'s identifier $ID_o \in \{0,1\}^*$, the KGC generates the partial private key for data owner $\mathcal{O}$ with the master key λ:

1) Compute $Q_o = H_1(ID_o) \in \mathbb{G}_1$.
2) Output the partial private key $D_o = Q_o^{\lambda} \in \mathbb{G}_1$.

KeyGen. Given system parameters ($\mathbb{G}_1$, $\mathbb{G}_2$, e, P, P_T, P_1, ..., P_k, H_1, H_2), data owner $\mathcal{O}$ chooses a random $x_o \in \mathbb{Z}_p^*$ as her secret key, and computes $P_o = P^{x_o} \in \mathbb{G}_1$ as her public key.

Sign. Given block $\boldsymbol{m}_i = (m_{i,1}, ..., m_{i,k}) \in \mathbb{Z}_p^k$ and block identifier id_i, data owner $\mathcal{O}$ computes a signature using her partial private key D_o, secret key x_o and the public aggregated key $(P_1, ..., P_k)$ as follows:

1) Compute $V_i = H_2(ID_o||P_o||id_i) \cdot \prod_{l=1}^{k} P_l^{m_{i,l}} \in \mathbb{G}_1$.
2) Output the signature σ_i on block $\boldsymbol{m}_i$ and block identifier id_i as $\sigma_i = V_i^{x_o} \cdot D_o \in \mathbb{G}_1$.

After computing all the signatures $(\sigma_1, ..., \sigma_n)$ on data M, data owner $\mathcal{O}$ outsources data M and all the signatures to the cloud.

ProofGen. To audit the integrity of data M stored in the cloud, a public verifier first generates an auditing challenge as follows:

1) Randomly pick a c-element set $\mathcal{J}$ to locate the c selected blocks that will be checked in this auditing challenge, where $\mathcal{J}$ is a subset of set $[1, n]$ and n is the total number of blocks in data M.
2) Generate a random value $y_j \in \mathbb{Z}_q$, for $j \in \mathcal{J}$, where q is a much smaller prime than p.
3) Output and send an auditing challenge $\{(j, y_j)\}_{j \in \mathcal{J}}$ to the cloud.

After receiving auditing challenge $\{(j, y_j)\}_{j \in \mathcal{J}}$, the cloud outputs a proof of possession of data M with the signatures on data M. Specifically,

1) Compute $\mu_l = \sum_{j \in \mathcal{J}} y_j m_{j,l} \in \mathbb{Z}_p$, where $l \in [1, k]$.
2) Aggregate signatures on the c selected blocks as $\sigma^* = \prod_{j \in \mathcal{J}} \sigma_j^{y_j} \in \mathbb{G}_1$.
3) Return an auditing response $\{\sigma^*, \boldsymbol{\mu}, \{id_j\}_{j \in \mathcal{J}}\}$ to the public verifier, where $\boldsymbol{\mu} = (\mu_1, .., \mu_k)$.

ProofVerify. Given auditing response $\{\sigma^*, \boldsymbol{\mu}, \{id_j\}_{j \in \mathcal{J}}\}$, auditing challenge $\{(j, y_j)\}_{j \in \mathcal{J}}$, data owner $\mathcal{O}$'s identifier ID_o, public key P_o, and system parameters ($\mathbb{G}_1$, $\mathbb{G}_2$, e, P, P_T, P_1, ..., P_k, H_1, H_2), the public verifier checks the correctness of this auditing response as follows:

1) Compute $Q_o = H_1(ID_o) \in \mathbb{G}_1$.
2) Compute $W_j = H_2(ID_o||P_o||id_j) \in \mathbb{G}_1$, where $j \in \mathcal{J}$.
3) Verify the following equation

$$e(\sigma^*, P) \overset{?}{=} e(\prod_{j \in \mathcal{J}} Q_o^{y_j}, P_T) \cdot e(\prod_{j \in \mathcal{J}} W_j^{y_j} \cdot \prod_{l=1}^{k} P_l^{\mu_l}, P_o). \tag{2}$$

If the above equation holds, then this public verifier believes the integrity of data M is correct; otherwise, it does not.

Fig. 3. Details of Certificateless Public Auditing.

where $W_i = H_2(ID_s||P_s||id_i)$. Then, given the hash value h, which is computed as $H_2(ID_s||P_s||id'') \cdot W_1 \cdot W_2 / W_3 \cdot W_4$, it is easy to find a string Ω, such that $h = H_2(\Omega)$, where $\Omega = ID_s||P_s||id'$. Clearly, it contradicts to the assumption that H_2 is a one-way hash function. Therefore, HA-CLS is non-malleable. ■

V. Certificateless Public Auditing in the Cloud

A. Overview

In this section, we build the entire certificateless public auditing mechanism in the cloud based on our homomorphic authenticable certificateless signature scheme. With our mechanism, a public verifier is able to audit the correctness of cloud data outsourced by the data owner without managing certificates.

Signature Size. Another practical problem we need to consider during the design of our certificateless public auditing mechanism is the signature size. As we presented in HA-CLS, a signature of a block is an element of $\mathbb{G}_1$, which is the same size of a block. It means that the data owner needs to spend the same size of storage on signatures as the size of storage on data. Because the size of data in the cloud is generally very large and service providers apply the pay-as-you-go pricing model in the cloud, storing data with the same size of signatures will certainly double the charges incurred to the data owner.

Therefore, it is better if we can reduce the overhead of signatures, so that the data owner does not have to incur a large amount of costs to store signatures. By leveraging an aggregated method from previous work [4], we can reduce the size of the signature to $1/k$ of the size of a block, where k is the number of elements in each block.

More specifically, with the aggregated method, a block is described as $\boldsymbol{m}_i = (m_{i,1}, ..., m_{i,k}) \in \mathbb{Z}_p^k$ instead of $m_i \in \mathbb{Z}_p$, and V_i is computed as $V_i = H_2(ID_o||P_o||id_i) \cdot \prod_{l=1}^{k} P_l^{m_{i,l}}$, where $(P_1, ..., P_k) \in \mathbb{G}_1^k$ is called a *public aggregated key*, ID_o is the signer identifier of the data owner $\mathcal{O}$ and P_o is her public key. Because signature σ_i of this block is still an element of $\mathbb{G}_1$, the size of a signature σ_i is only $1/k$ of the size of block $\boldsymbol{m}_i$. As a necessary tradeoff, the computation and communication cost during public auditing will be higher, with an increase in the value of k.

B. Design of Certificateless Public Auditing

Our certificateless public auditing mechanism includes six algorithms, **Setup**, **Partial-Private-Key-Extract**, **KeyGen**, **Sign**, **ProofGen** and **ProofVerify**. Similar as HA-CLS, by running **Setup**, **Partial-Private-Key-Extract**, **KeyGen** and **Sign**, the data owner is able to obtain her partial private key, secret key and public key, and compute signatures on blocks. In **ProofGen**, the cloud is able to generate a proof of possession of data. In **ProofVerify**, a public verifier is able to check the correctness of the proof before utilizing cloud data. Details of these algorithms are described in Fig. 3.

Discussion. To protect data privacy at the same time, some certificateless public key encryption mechanisms (such as [27]) on cloud data can be used. Further details can be found in [27]. The main objective of this paper is to design a certificateless public auditing mechanism to maintain data integrity.

C. Security Analysis of Certificateless Public Auditing

Now, we analyze the security of our certificateless public auditing mechanism, including correctness and unforgeability.

***Theorem** 3: Given an auditing response $\{\sigma^*, \boldsymbol{\mu}, \{id_j\}_{j\in\mathcal{J}}\}$, data owner $\mathcal{O}$'s identifier ID_o, public key P_o, and system parameters ($\mathbb{G}_1$, $\mathbb{G}_2$, e, P, P_T, P_1, ..., P_k H_1, H_2), a public verifier is able to correctly check the integrity of data M.*

Proof: Based on the correctness of Equation (1) and (3), the correctness of Equation (2) can be provesd as follows:

$$\begin{aligned}
& e(\prod_{j\in\mathcal{J}} Q_o^{y_j}, P_T)\cdot e(\prod_{j\in\mathcal{J}} W_j^{y_j}\cdot\prod_{l=1}^{k} P_l^{\mu_l}, P_o) \\
= \;& e(\prod_{j\in\mathcal{J}} Q_o^{y_i}, P^{\lambda})\cdot e(\prod_{j\in\mathcal{J}} W_j^{y_j}\cdot\prod_{l=1}^{k} P_l^{\sum_{j\in\mathcal{J}} y_j m_{j,l}}, P^{x_o}) \\
= \;& e(\prod_{j\in\mathcal{J}} (Q_o^{\lambda})^{y_i}, P)\cdot e(\prod_{j\in\mathcal{J}} W_j^{y_j}\cdot\prod_{l=1}^{k}\prod_{j\in\mathcal{J}} P_l^{y_j m_{j,l}}, P^{x_o}) \\
= \;& e(\prod_{j\in\mathcal{J}} D_o^{y_i}, P)\cdot e(\prod_{j\in\mathcal{J}} W_j^{y_j}\cdot\prod_{j\in\mathcal{J}}(\prod_{l=1}^{k} P_l^{m_{j,l}})^{y_j}, P^{x_o}) \\
= \;& e(\prod_{j\in\mathcal{J}} D_o^{y_i}, P)\cdot e(\prod_{j\in\mathcal{J}} (W_j\cdot\prod_{l=1}^{k} P_l^{m_{j,l}})^{y_j}, P^{x_o}) \\
= \;& e(\prod_{j\in\mathcal{J}} D_o^{y_i}, P)\cdot e(\prod_{j\in\mathcal{J}} (V_j^{x_o})^{y_j}, P) \\
= \;& e(\prod_{j\in\mathcal{J}} \sigma_j^{y_j}, P) = e(\sigma^*, P).
\end{aligned}$$

Therefore, a public verifier is able to correctly audit data integrity. ■

***Theorem** 4: For an untrusted cloud, it is computationally infeasible to generate a forgery of an auditing response with our mechanism.*

Proof: To generate a forgery of an auditing response, an untrusted cloud can operate in the two following ways.

First, it generates a forgery of a signature on each block, then it computes a forgery of an auditing response based on the forgeries of the signatures on all the blocks. However, as we proved in Theorem 1, for an entity, who does not have the private key of data owner $\mathcal{O}$, it is computationally infeasible to generate a forgery of a signature because solving the CDH problem in $\mathbb{G}_1$ is hard.

Second, without generating any forgery of a signature, the untrusted cloud directly generates a forgery of an auditing response on corrupted data M' by winning a game, which is denoted as Game 1. Following the security model in [4], we define the game as follows:

Game 1: A public verifier sends an auditing response $\{(j, y_j)\}_{j\in\mathcal{J}}$ to the cloud, the auditing response on the correct data M should be $\{\sigma^*, \boldsymbol{\mu}, \{id_j\}_{j\in\mathcal{J}}\}$. Instead of generating the correct auditing response, the untrusted cloud generates a forgery of an auditing response on corrupted data M' as $\{\sigma^*, \boldsymbol{\mu}', \{id_j\}_{j\in\mathcal{J}}\}$, where $\boldsymbol{\mu}' = (\mu_1', ..., \mu_k')$, $\mu_l' = \sum_{j\in\mathcal{J}} y_j m_{j,l}'$, for $1 \le l \le k$, and $m_{j,l}' \in M'$. Define $\Delta\mu = \mu_l' - \mu_l$, for $1 \le l \le k$, and at least one element of $\{\Delta\mu_l\}_{1\le l\le k}$ is nonzero since $M \ne M'$. If this forgery on corrupted data M' can successfully pass the verification performed by the public verifier, then the untrusted cloud wins the game. Otherwise, it loses.

Now, we assume that the untrusted cloud could win the game above, which means $\{\sigma^*, \boldsymbol{\mu}', \{id_j\}_{j\in\mathcal{J}}\}$ successfully passes the verification, then we have

$$e(\sigma^*, P) = e(\prod_{j\in\mathcal{J}} Q_o^{y_j}, P_T)\cdot e(\prod_{j\in\mathcal{J}} W_j^{y_j}\cdot\prod_{l=1}^{k} P_l^{\mu_l'}, P_o).$$

According to the correct auditing response $\{\sigma^*, \boldsymbol{\mu}, \{id_j\}_{j\in\mathcal{J}}\}$ on correct data M, we also have

$$e(\sigma^*, P) = e(\prod_{j\in\mathcal{J}} Q_o^{y_j}, P_T)\cdot e(\prod_{j\in\mathcal{J}} W_j^{y_j}\cdot\prod_{l=1}^{k} P_l^{\mu_l}, P_o).$$

Clearly, we can learn that

$$\prod_{l=1}^{k} P_l^{\mu_l} = \prod_{l=1}^{k} P_l^{\mu_l'}, \quad \prod_{l=1}^{k} P_l^{\Delta\mu_l} = 1.$$

Because $\mathbb{G}_1$ is a cyclic group, for two random elements A, $B \in \mathbb{G}_1$, there exists $x \in \mathbb{Z}_p$ so that $A = B^x$. Without loss of generality, given A, B, each P_l can be randomly generated as $P_l = A^{\xi_l} B^{\gamma_l}$, where ξ_l and γ_l are random values of $\mathbb{Z}_p$. Then, we learn that

$$1 = \prod_{l=1}^{k} P_l^{\Delta\mu_l} = \prod_{l=1}^{k} (A^{\xi_l} B^{\gamma_l})^{\Delta\mu_l} = A^{\sum_{l=1}^{k}\xi_l\Delta\mu_l}.B^{\sum_{l=1}^{k}\gamma_l\Delta\mu_l}.$$

Clearly, we can find a solution of the Discrete Logarithm problem with a probability of $1 - 1/p$. More specifically, given B, $A = B^x \in \mathbb{G}_1$, we can output

$$A = B^{-\frac{\sum_{l=1}^{k}\gamma_l\Delta\mu_l}{\sum_{l=1}^{k}\xi_l\Delta\mu_l}}, \quad x = -\frac{\sum_{l=1}^{k}\gamma_l\Delta\mu_l}{\sum_{l=1}^{k}\xi_l\Delta\mu_l}.$$

unless the denominator $\sum_{l=1}^{k}\xi_l\Delta\mu_l$ is zero.

However, as we defined in Game 1, at least one element of $\{\Delta\mu_l\}_{1\le l\le k}$ is nonzero, and ξ_l is a random element of $\mathbb{Z}_p$, therefore, the denominator is nonzero with probability of $1 - 1/p$. It means, if the untrusted cloud could win the game, then we can find a solution of the Discrete Logarithm problem with a probability of $1 - 1/p$, which contradicts to the assumption that the Discrete Logarithm problem is hard in $\mathbb{G}_1$. Therefore, it is computationally infeasible to generate a forgery of an auditing response with our certificateless public auditing mechanism. ■

VI. PERFORMANCE

In this section, we first analyze the computation and communication cost of our certificateless public auditing mechanism, and then evaluate the performance of our mechanism.

TABLE I
COMPARISON OF COMPUTATION COST IN **SIGN**, **PROOFGEN** AND **PROOFVERIFY**

	Certificate-based Mechanism [4]	Our Certificateless Mechanism
Sign	$(k+1)\mathtt{Exp}_{\mathbb{G}_1} + k\mathtt{Mul}_{\mathbb{G}_1} + \mathtt{Hash}_{\mathbb{G}_1}$	$(k+1)\mathtt{Exp}_{\mathbb{G}_1} + (k+1)\mathtt{Mul}_{\mathbb{G}_1} + \mathtt{Hash}_{\mathbb{G}_1}$
ProofGen	$c\mathtt{Exp}_{\mathbb{G}_1} + c\mathtt{Mul}_{\mathbb{G}_1}$	$c\mathtt{Exp}_{\mathbb{G}_1} + c\mathtt{Mul}_{\mathbb{G}_1}$
ProofVerify	$2\mathtt{Pair} + (c+k)\mathtt{Exp}_{\mathbb{G}_1} + (c+k)\mathtt{Mul}_{\mathbb{G}_1} + c\mathtt{Hash}_{\mathbb{G}_1}$	$3\mathtt{Pair} + (2c+k)\mathtt{Exp}_{\mathbb{G}_1} + (2c+k)\mathtt{Mul}_{\mathbb{G}_1} + \mathtt{Mul}_{\mathbb{G}_2} + c\mathtt{Hash}_{\mathbb{G}_1}$

A. Computation Cost

According to algorithm **Sign** in Section V, the computation cost of calculating a signature is about $(k+1)\mathtt{Exp}_{\mathbb{G}_1} + (k+1)\mathtt{Mul}_{\mathbb{G}_1} + \mathtt{Hash}_{\mathbb{G}_1}$, where $\mathtt{Exp}_{\mathbb{G}_1}$ denotes the cost of computing one exponentiation in $\mathbb{G}_1$, $\mathtt{Mul}_{\mathbb{G}_1}$ denotes the cost of computing one multiplication in $\mathbb{G}_1$, and $\mathtt{Hash}_{\mathbb{G}_1}$ denotes the cost of computing one hashing operation in $\mathbb{G}_1$.

As described in algorithm **ProofGen** and **ProofVerify**, the computation cost of generating an auditing response is about $c\mathtt{Exp}_{\mathbb{G}_1} + c\mathtt{Mul}_{\mathbb{G}_1}$, and the computation cost of verifying an auditing response is about $3\mathtt{Pair} + (2c+k)\mathtt{Exp}_{\mathbb{G}_1} + (2c+k)\mathtt{Mul}_{\mathbb{G}_1} + \mathtt{Mul}_{\mathbb{G}_2} + c\mathtt{Hash}_{\mathbb{G}_1}$, where $\mathtt{Pair}$ denotes the cost of computing one pairing operation on $\mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_2$ and $\mathtt{Mul}_{\mathbb{G}_2}$ denotes the cost of computing one multiplication in $\mathbb{G}_2$.

Compared to a certificate-based public auditing mechanism [4], which is the state of the art and also built based on bilinear maps, our certificateless public auditing mechanism has the same computation cost in **Sign** and **ProofGen**, but requires more computation cost in **ProofVerify**, which is the extra cost introduced by avoiding managing certificates in a public auditing mechanism. A detailed comparison of computation cost between this certificate-based mechanism [4] and our mechanism is illustrated in Table I.

B. Communication Cost

To check the integrity of data in the cloud, a public verifier first needs to send an auditing challenge $\{(j, y_j)\}_{j \in \mathcal{J}}$ to the cloud, and then the cloud needs to send an auditing response $\{S^*, \boldsymbol{\mu}, \{id_j\}_{j \in \mathcal{J}}\}$ back to the public verifier. The communication cost of an auditing challenge is $c(|q|+|n|)$ bits, and the communication cost of an auditing response is $(k+1)|p|+c|id|$ bits, where $|q|$ is the length of an element of $\mathbb{Z}_q$, $|p|$ is the length of an element of $\mathbb{Z}_p$, n is the total number of blocks in data and $|id|$ is the length of a block identifier. Compared to the communication cost in the certificate-based mechanism [4], our mechanism requires the same communication cost.

C. Experimental Results

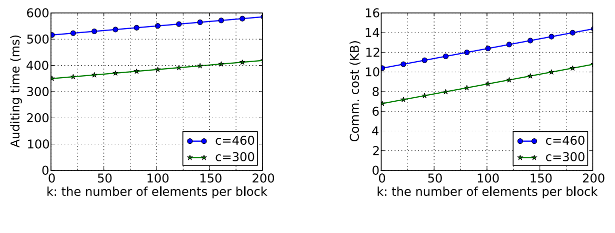
We now evaluate the computation and communication cost experimentally. In the following experiments, we leverage the Pairing Based Cryptography (PBC) library to implement cryptographic operations. All the experiments are tested using a Mac OS X system with a 1.83 GHz Intel Core Duo processor and 2 GB 667 MHz DDR2 memory.

We assume the total number of blocks in data is $n = 1,000,000$, $|p| = 160$ bits and $k = 100$, then the size of entire data is 2 GB. In addition, we set $|q| = 80$ bits, $|n| = 20$ bits and $|id| = 80$ bits as in our recent work [12], and choose an MNT curve with a base field size of 159 bits. According to previous work [2], when 1% of all the blocks are polluted, a public verifier can keep the detection probability greater than 99% by choosing $c = 460$ random blocks. If a smaller number of random blocks is selected, then a public verifier can finish the auditing in a shorter period of time; however, as a tradeoff, the detection probability will decrease. For instance, if $c = 300$, the detection probability is only greater than 95%.

TABLE II
COMPARISON OF AUDITING PERFORMANCE

	Certificate-based [4]		Our Mechanism	
Selected Blocks	460	300	460	300
Auditing Time (ms)	391.07	280.36	569.31	403.23
Commun. Cost (KB)	12.37	8.77	12.37	8.77

Based on our analysis of computation and communication cost, we compare the auditing performance between the certificate-based mechanism [4] and our mechanism in Table II. We find that, both mechanisms are able to allow a public verifier to check the integrity of data without retrieving the entire data from the cloud. Compared to the certificate-based solution, our mechanism requires more auditing time to finish the verification on the same auditing response. More specifically, if $c = 460$, our mechanism requires about 569.31 milliseconds to verify the correctness of data while the certificate-based mechanism only needs about 391.07 milliseconds. However, since our mechanism is able to avoid asking verifiers to manage certificates, it can successfully eliminate the security risks introduced in the certificate-based mechanism.



(a) Impact of k on auditing time (ms) (b) Impact of k on comm. cost (KB)

Fig. 4. Impact of k on the auditing performance.

In Fig. 4(a), we can see that the auditing times with different numbers of selected blocks are both linearly increasing with the number of elements in each block. It is because an increase in the value of k will linearly introduce more exponentiations and multiplications in $\mathbb{G}_1$ during the verification of an auditing response. In addition, as shown in Fig. 4(b), the increase in the value of k will also increase the communication cost.

VII. RELATED WORK

Public Auditing. Ateniese *et al.* [2] proposed provable data possession (PDP), which enables a user to verify the integrity of data stored in an untrusted server without downloading the entire data. This mechanism is the first one that supports public auditing. Shacham and Waters [4] designed an improved public auditing mechanism based on BLS signatures. Juels and Kaliski [21] defined another similar model named proof of retrievability (POR), which is also able to verify the integrity of data in an untrusted server. In this mechanism, the user verifies the integrity of data by asking the server to return the values of *sentinels*, which are special blocks and randomly added in the original file by the user.

To support dynamic data, Wang *et al.* [6] utilized the Merkle Hash Tree during the design of a public auditing mechanism. Rank-based authenticated dictionary [7] and index hash tables [10] can also be used to support dynamic data. In addition, how to audit the integrity of data without downloading the entire data, where data is encoded with erasure codes [5], network coding [9] and LT codes [11], are also studied in previous works. Wang *et al.* [8] considered data privacy under public auditing. With the usage of random maskings, a public verifier in their mechanism is able to check the integrity of cloud data but cannot obtain any private data. Our recent works [12], [15], [16], [28] focus on preserving identity privacy from a public verifier for a group of users when auditing the integrity of shared data in the cloud. In addition, we also designed a public auditing mechanism [14] to support efficient user revocation on cloud shared data by taking advantage of proxy re-signatures.

Certificateless Signatures. Al-Riyami and Paterson first proposed Certificateless Public Key Cryptography [19], which does not need certificates as in PKI and avoids the inherent key escrow problem in Identity-based Public Key Cryptography [22]. The following works focus on different constructions based on bilinear maps [25] and improving the security of certificateless signatures [24]. More recently, Zhang *et al.* [26] proposed a certificateless aggregate signature scheme, which allows different signatures on different messages being compressed into one, however, those corresponding messages cannot be compressed. Therefore, none of them can be directly utilized into a public auditing mechanism for efficiently verifying data integrity in the cloud.

VIII. Conclusion

In this paper, we propose the first certificateless public auditing mechanism for verifying data integrity in an untrusted cloud. With our mechanism, a public verifier is not only able to audit data integrity in the cloud but also able to eliminate possible security risks introduced by PKI in previous solutions. We proved that the security of our proposed mechanism is based on the CDH assumption and DL assumption. Experimental results show that our mechanism is efficient.

Acknowledgement

We are grateful to the anonymous reviewers for their helpful comments. This work is supported by the NSF of China (No. 61272457 and 61170251), Fundamental Research Funds for the Central Universities (No. K50511010001), National 111 Program (No. B08038), Doctoral Foundation of Ministry of Education of China (No. 20100203110002), Program for Changjiang Scholars and Innovative Research Team in University (PCSIRT 1078).

References

[1] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "A View of Cloud Computing," *Communications of the ACM*, vol. 53, no. 4, pp. 50–58, April 2010.

[2] G. Ateniese, R. Burns, R. Curtmola, J. Herring, L. Kissner, Z. Peterson, and D. Song, "Provable Data Possession at Untrusted Stores," in *the Proceedings of ACM CCS 2007*, 2007, pp. 598–610.

[3] (2011, June) Yesterday's Authentication Bug. [Online]. Available: http://blog.dropbox.com/index.php/yesterdays-authentication-bug/

[4] H. Shacham and B. Waters, "Compact Proofs of Retrievability," in *the Proceedings of ASIACRYPT 2008*. Springer-Verlag, 2008, pp. 90–107.

[5] C. Wang, Q. Wang, K. Ren, and W. Lou, "Ensuring Data Storage Security in Cloud Computing," in *the Proceedings of IEEE/ACM IWQoS 2009*, 2009, pp. 1–9.

[6] Q. Wang, C. Wang, J. Li, K. Ren, and W. Lou, "Enabling Public Verifiability and Data Dynamic for Storage Security in Cloud Computing," in *the Proceedings of ESORICS 2009*. Springer-Verlag, 2009, pp. 355–370.

[7] C. Erway, A. Kupcu, C. Papamanthou, and R. Tamassia, "Dynamic Provable Data Possession," in *the Proceedings of ACM CCS 2009*, 2009, pp. 213–222.

[8] C. Wang, Q. Wang, K. Ren, and W. Lou, "Privacy-Preserving Public Auditing for Data Storage Security in Cloud Computing," in *the Proceedings of IEEE INFOCOM 2010*, 2010, pp. 525–533.

[9] B. Chen, R. Curtmola, G. Ateniese, and R. Burns, "Remote Data Checking for Network Coding-based Distributed Stroage Systems," in *the Proceedings of ACM CCSW 2010*, 2010, pp. 31–42.

[10] Y. Zhu, H. Wang, Z. Hu, G.-J. Ahn, H. Hu, and S. S.Yau, "Dynamic Audit Services for Integrity Verification of Outsourced Storage in Clouds," in *the Proceedings of ACM SAC 2011*, 2011, pp. 1550–1557.

[11] N. Cao, S. Yu, Z. Yang, W. Lou, and Y. T. Hou, "LT Codes-based Secure and Reliable Cloud Storage Service," in *the Proceedings of IEEE INFOCOM 2012*, 2012, pp. 693–701.

[12] B. Wang, B. Li, and H. Li, "Oruta: Privacy-Preserving Public Auditing for Shared Data in the Cloud," in *the Proceedings of IEEE Cloud 2012*, June 2012.

[13] A. Juels and A. Oprea, "New Approaches to Security and Availability for Cloud Data," *Communications of the ACM*, vol. 56, no. 2, pp. 64–73, 2013.

[14] B. Wang, B. Li, and H. Li, "Public Auditing for Shared Data with Efficient User Revoation in the Cloud," in *the Proceedings of IEEE INFOCOM 2013*, 2013.

[15] B. Wang, H. Li, and M. Li, "Privacy-Preserving Public Auditing for Shared Cloud Data Supporting Group Dynamics," in *the Proceedings of IEEE ICC 2013*, 2013.

[16] B. Wang, S. S. Chow, M. Li, and H. Li, "Storing Shared Data on the Cloud via Security-Mediator," in *the Proceedings of ICDCS'13*, 2013.

[17] C. Ellison and B. Schneier, "Ten Risks of PKI: What You're Not Being Told About Public Key Infrastructure," *Computer Security Journal*, vol. 16, no. 1, pp. 1–7, 2000.

[18] P. Gutmann, "PKI: It's Not Dead, Just Resting," *IEEE Computer*, vol. 35, no. 8, pp. 41–49, 2002.

[19] S. S. Al-Riyami and K. G. Paterson, "Certificateless Public Key Cryptography," in *the Proceedings of ASIACRYPT 2003*. Springer-Verlag, 2003, pp. 452–473.

[20] G. Ateniese, R. D. Pietro, L. V. Mancini, and G. Tsudik, "Scalable and Efficient Provable Data Possession," in *the Proceedings of the 4th International Conference on Security and Privacy in Communication Networks (SecureComm 2004)*, 2008.

[21] A. Juels and B. S. Kaliski, "PORs: Proofs of Retrievability for Large Files," in *the Proceedings of ACM CCS 2007*, 2007, pp. 584–597.

[22] A. Shamir, "Identity-based Cryptosystems and Signature Schemes," in *the Proceedings of CRYPTO'84*. Springer-Verlag, 1984, pp. 47–53.

[23] J. C. Cha and J. H. Cheon, "An Identity-Based Signature from Gap Diffie-Hellman Groups," in *the Proceedings of PKC 2003*. Springer-Verlag, 2003, pp. 18–30.

[24] X. Huang, W. Susilo, Y. Mu, and F. Zhang, "On the Security of Certificateless Signature Schemes from Asiacrypt 2003," in *the Proceedings of CANS 2005*. Springer-Verlag, 2005, pp. 13–25.

[25] Z. Zhang, D. S. Wong, J. Xu, and D. Feng, "Certificateless Public-Key Signature: Security Model and Efficient Construction," in *the Proceedings of ACNS 2006*. Springer-Verlag, 2006, pp. 293–308.

[26] L. Zhang and F. Zhang, "A New Certificateless Aggregate Signature Scheme," *Computer Communications*, vol. 32, no. 6, pp. 1079–1085, 2009.

[27] L. Xu, X. Wu, and X. Zhang, "CL-PRE: a Certificateless Proxy Re-Encryption Scheme for Secure Data Sharing with Public Cloud," in *the Proceedings of ACM ASIACCS 2012*, 2012.

[28] B. Wang, B. Li, and H. Li, "Knox: Privacy-Preserving Auditing for Shared Data with Large Groups in the Cloud," in *the Proceedings of ACNS 2012*. Springer-Verlag, June 2012, pp. 507–525.