Journal of Circuits, Systems, and Computers
Vol. 27, No. 11 (2018) 1850181 (17 pages)

DOI: 10.1142/S0218126618501815

World Scientific
www.worldscientific.com

Certificateless Public Verification for the Outsourced Data Integrity in Cloud Storage*

Longxia Huang, Junlong Zhou†, Gongxuan Zhang‡, Jin Sun, Tian Wang and Ahmadreza Vajdi

School of Computer Science and Engineering,
Nanjing University of Science and Technology,
Nanjing 210094, P. R. China
†*jlzhou@njust.edu.cn*
‡*gongxuan@njust.edu.cn*

Received 19 September 2017
Accepted 18 December 2017
Published 15 February 2018

By advances in cloud storage systems, users have access to the data saved in the cloud and can manipulate the data without limitation of time and place. As the data owner no longer possesses data physically, he is required to ensure the integrity of the data stored in the cloud with the public key given by public key infrastructure (PKI). Thus the security of PKI and certificates are essential. However, there are numerous security risks in the traditional PKI and it is complex to administer the certificates. Certificateless public key cryptography is used in this paper to solve these problems. We also use elliptic curve group to reduce computation overhead. In this paper, we design a certificateless public verification mechanism to check the integrity of data outsourced in the cloud and we further extend it to support a multiuser group by batch verification. Specifically, a public verifier who replaces the data owner to check the integrity in the proposed scheme does not require to manage any certificates during the verification process. Meanwhile, a verifier is not required to download the entire file for integrity checking. Theoretical analyses verify the security of our scheme and experimental results show its efficiency.

Keywords: Certificateless public verification; cloud storage; data security.

1. Introduction

With the rapid development of cloud computing, as a part of cloud computing, cloud storage[1] which manages and organizes data, allows data users to outsource and have access to their files anytime, with any device and from anywhere. Thus users prefer to store data in the cloud rather than store them locally. On the one hand, cloud storage greatly reduces users' pressure on storage and calculation,[2] and every group member is able to access the shared data.[3] On the other hand, it offers scalable payment which

*This paper was recommended by Regional Editor Tongquan Wei.
‡Corresponding author.

costs less than storing the data locally.[4] However, the cloud server is not reliable[5] enough to neglect some security problems in cloud storage. First, the data stored in the cloud may be spoiled because of hardware failures or human errors. Second, the data security was threatened by attacks inside and outside. What is worse, to avoid the loss of reputation or profits, cloud server may even conceal accidents of data errors. Therefore, it is an urgent security demand to verify the integrity of shared data stored in the cloud.

Digital signatures and hash functions are utilized to handle the problem of cloud data integrity checking. In 2007, Ateniese *et al.*[6] first proposed provable data possession (PDP) and formalized the security model for this primitive. To enable cloud users to check the integrity of their outsourced data without knowing the entire file efficiently, they presented two highly effective and provably secure PDP schemes.[6,7] At the same time, Juels and Kaliski[8] proposed the concept of proof of retrievability (PoR) which enables the cloud server to generate a correct proof that the cloud user can retrieve the remote file. In PoR, they used pseudorandom sampling technique to present a sentinel-based construction. However, the number of sentinels limits the times of verification. All the approaches[6–8] need a user with the private key to realize integrity verification.[9,10]

To release users' pressure on data integrity verification, a third-party auditor (TPA) is introduced to verify the integrity of files stored in the cloud instead of users. However, during the verification, the verifier may get some confidential message from the proof message received from the cloud. Therefore, privacy protection[11–15] should be taken into account in public auditing protocol. Wang *et al.*[11] constructed homomorphic authenticator scheme based on ring signature to protect the identity of file signer. By integrating public key-based homomorphic linear authenticator with random masking, Wang *et al.*[14] proposed a protocol which guarantees that the TPA could not learn any knowledge about the data content stored in the cloud server during the efficient auditing process. Yu *et al.*[16] also applied "zero-knowledge privacy" to ensure that the verifier cannot obtain any information from public available data. Besides data confidentiality, Hwang *et al.*'s scheme[17] also supports data dynamic operations and batch auditing, but the security proof is heuristic and they did not show experimental results in the paper. As these schemes[11–15,17] are all based on public key infrastructure (PKI), they all have some drawbacks such as expensive computation cost and complex certificate management.

To address the key management issues, some protocols[18–21] based on identity cryptography have been put forward. For example, Zhang and Dong[21] proposed an efficient identity-based auditing protocol for cloud data integrity and extended their protocol to assist multiuser setting. However, as Identity-based signature[22] has key escrow problem and certificate management problem, it is better to use certificateless signature (CLS) to construct public auditing scheme. Wang *et al.*[23] proposed the CPA scheme which is the first certificateless[24] public auditing mechanism for verifying data integrity in the cloud. Then, CLPA was proposed by He *et al.*[25] to

support wireless body area networks. But these two schemes[23,25] cannot support multiuser groups.

In this paper, we first design a certificateless signature scheme based on the technology of elliptic curve and bilinear maps. Then, we further build an entire certificateless public verification mechanism for verifying the integrity of outsourced data in an unreliable cloud. Furthermore, we extend our scheme to support multiuser in cloud storage to make the public auditing more efficient. Public verifiers in our certificateless public verification scheme do not need to manage certificates, which eliminates the security risks in schemes using PKI. In addition, our protocol guarantees a verifier to check the correctness of the outsourced data efficiently without learning any knowledge about the data content stored in the cloud server during the efficient auditing process. Specifically, our contribution can be summarized as follows.

(i) We first present a certificateless signature which uses the knowledge of the elliptic curve and bilinear maps. We then propose a public verification scheme which can be used to check the cloud data integrity based on the signature.
(ii) Our scheme supports scalable and efficient public verification for data stored in the cloud. Specifically, it constructs a batch auditing approach which can be carried out by the verifier to breeze through auditing tasks from different users simultaneously. The experimental results show that the auditing overhead is independent of the group user numbers.
(iii) We prove that the proposed scheme is provably secure under the discrete logarithm assumption (DLA) and the experimental results show that our scheme is more efficient as compared to representative methods.

The rest of the paper is organized as follows: Sec. 2 introduces the cryptographic primitives. Then, we provide the detailed description of our model system and a certificateless signature in Sec. 3. In Sec. 4, we describe a certificateless public verification mechanism for verifying outsourced data in an unreliable cloud based on the certificateless signature. We design a public auditing scheme which satisfies a multiuser group in Sec. 5, followed by Sec. 6 where the security and performance of our work are evaluated. Finally, Sec. 7 gives the concluding remark of this paper.

2. Preliminaries

In this section, we describe three kinds of technology to make the readers feel easier to understand our scheme.

2.1. *Bilinear map*

Let G_1 and G_2 be two multiplicative cyclic groups of prime order q. Let g be a generator of G_1. A bilinear map is a map $e : G_1 \times G_1 \to G_2$. This bilinearity implies that for any $u_1, u_2, v \in G_1$, we have $e(u_1 \cdot u_2, v) = e(u_1, v) \cdot e(u_2, v)$. Of course, there

is an efficiently computable algorithm for computing e and the map should be nontrivial, i.e., e is nondegenerate: $e(g, g) \neq 1$.

2.2. *Elliptic curve group*

Let E/F denote an elliptic curve E over a prime finite field F, the points in it satisfied the following equation: $y^2 = x^3 + ax + ba,\ b \in F$ with the discriminant $\Delta = 4a^3 + 27b^2 \neq 0$. An extra point O called the point at infinity together with the points on E/F form a group $G = \{(x, y) : x, y \in F, E(x, y) = 0\} \cup \{O\}$.

In polynomial time, we assume two problems viz. discrete logarithm problem (DLP) and DLA, which are defined in group G to be intractable:

- DLP: The discrete logarithm problem states that it is hard to compute x for a party who knows the generator P of G and $Q = x \cdot P$.
- DLA: The discrete logarithm assumption states that for any polynomial-time algorithm, the probability of solving the DLP is negligible.

2.3. *Certificateless signatures*

In 2003, Al-Riyami and Paterson[24] proposed certificateless signatures to release entities from managing certificates in public key cryptography. Key escrow problem which exists in identity-based Signatures, is not in CLSs. A CLS scheme consists of five algorithms as follows:

(1) *Setup*: With a security parameter, this algorithm sets up a master key and the system parameters.
(2) *Partial-private-extract*: A user gets a part of his secret key and registers his identity to key generation center (KGC) to get the remaining part.
(3) *KeyGen*: KGC returns the identity-based private key to the user through a secure channel. The user gets his key pair after ensuring the correctness.
(4) *Sign*: A user signs a message with his secret key and the public parameters.
(5) *Verify*: The correctness of the signature is proven by passing an equation.

In CLS scheme, the KGC only generates a partial private key of an entity and the entity generates the remaining part of his private key himself. Therefore, the KGC does not have the whole knowledge of an entity's private key and it does not have the ability to compute a valid signature. Al-Riyami and Paterson.[24] defined that a secure CLS scheme is able to resist two kinds of adversaries, $\mathcal{A}1$ as a malicious KGC and $\mathcal{A}2$ as a dishonest user.

Adversary 1: Adversary $\mathcal{A}1$ can access to the master key, but cannot replace any user's public key.

Adversary 2: Adversary $\mathcal{A}2$ does not have access to the master key, but $\mathcal{A}2$ can replace user's public key as our scheme is certificateless.

3. Model and Signature Scheme

In this section, we first introduce the system model, and then show the design of signature scheme.

3.1. *System model*

For a certificateless public verification system in cloud data storage, the architecture is illustrated in Fig. 1. Our system model consists of four entities: a data user group, a public verifier, the cloud server and a KGC.

A data user group consists of a lot of users who have large amount of data files to save or share. A public verifier is a server with the ability of computation and he/she is able to check the integrity of data outsourced by a challenge-and-response protocol. The cloud server has significant storage space and computation resources. A KGC will generate a partial secret key of a user who has registered his/her identity before. In the verification, a data owner sends verify request to public verifier to examine the integrity of the outsourced data. Upon receiving this request, the verifier generates a challenge message and sends it to the cloud server. Then the cloud server

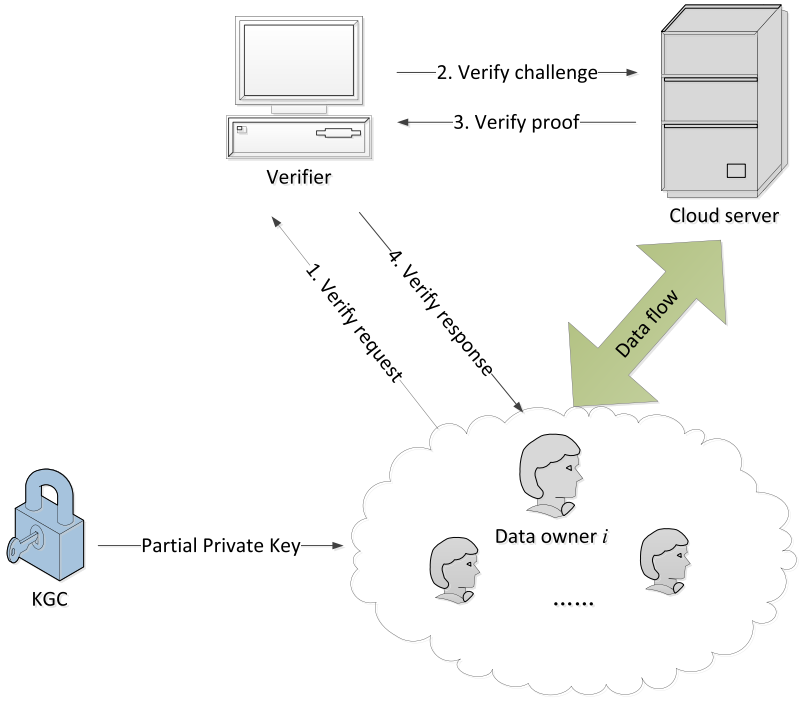

Fig. 1. System model.

replies the proof message to the verifier. Finally, the public verifier checks the correctness of the proof message and returns 0/1 to the data owner.

3.2. *Signature scheme*

Our signature scheme consists of five parts: *Setup*, *Partial-private-extract*, *KeyGen*, *Sign* and *Verify*. The details of these algorithms are described as follows.

Setup. Given a security parameter k, this algorithm sets up a master key and system parameters. Upon getting k, KGC has four steps to do as follows. First, it selects an additive cyclic group G_1 and a multiplicative G_2 of the same prime order $q > 2k$. Second, it chooses $e : G_1 \times G_1 \to G_2$ as a bilinear map and P, T as two generators of group G_1. There are two hash functions, which are used to map data of arbitrary size to data of fixed size, $H_1 : \{0,1\}^* \to G_1$ and $H_2 : G_1 \times \{0,1\}^* \to Z_q$. Then, KGC randomly takes $x \in Z_q$ as its master private key and computes its public key as $P_{\text{pub}} = x \cdot P$. Finally, it keeps master private key x secretly and publishes system parameters as $\text{Param} = (G_1, G_2, q, e, P, T, H_1, H_2, P_{\text{pub}})$.

Partial-private-extract. The user i with identity ID randomly picks x_{ID} as part of his secret key and computes $P_{\text{ID}} = x_{\text{ID}} \cdot P$. Then, he registers his identity to KGC and gets the rest of his secret key.

KeyGen. With a user's identifier ID, the master key x, user's public key P_{ID} and system parameters, this algorithm returns the user's identity-based private key. In this step, KGC chooses a random r_{ID}, computes $R_{\text{ID}} = r_{\text{ID}} \cdot P$ and $h_{\text{ID}} = H_1(\text{ID}, R_{\text{ID}}, P_{\text{ID}})$. Then KGC computes $s_{\text{ID}} = r_{\text{ID}} + h_{\text{ID}} \cdot x$ and sends $(s_{\text{ID}}, R_{\text{ID}})$ to the user through a secure channel.

After getting $(s_{\text{ID}}, R_{\text{ID}})$, the user needs to check the validity of s_{ID} through verifying the equality $s_{\text{ID}}P = R_{\text{ID}} + h_{\text{ID}} \cdot P_{\text{pub}}$. The user will accept s_{ID} if the equation holds. Because $s_{\text{ID}} = r_{\text{ID}} + h_{\text{ID}} \cdot x$, $R_{\text{ID}} = r_{\text{ID}} \cdot P$ and $x_{\text{ID}} \cdot P = P_{\text{ID}}$, we can find that the equation holds. Finally, the private key of user with identity ID is the pair $\text{sk}_{\text{ID}} = (s_{\text{ID}}, x_{\text{ID}})$ and the public key is $\text{pk}_{\text{ID}} = (P_{\text{ID}}, R_{\text{ID}})$.

Sign. The user with identity ID takes the system parameters $\text{Param} = (G_1, G_2, q, e, P, T, H_1, H_2, P_{\text{pub}})$, his own private key $\text{sk}_{\text{ID}} = (s_{\text{ID}}, x_{\text{ID}})$ and a message m with abstract index as inputs, then generates a signature of the message. In this algorithm, the user computes $h = H_2(m||\text{index}||\text{ID})$ and computes its signature as $s = x_{\text{ID}} \cdot m \cdot T + h \cdot s_{\text{ID}}$. The user uploads the signature and the message to the cloud server.

Verify. After receiving the signature and message, the cloud server needs to verify the correctness of the signature. The verifier first computes $h_{\text{ID}} = H_1(\text{ID}, R_{\text{ID}}, P_{\text{ID}})$, and then verifies whether $e(s, P) = e(T, m \cdot P_{\text{ID}}) \cdot e(h, R_{\text{ID}}) \cdot e(h \cdot h_{\text{ID}}, P_{\text{pub}})$ holds.

Signature correctness means that a valid signature is able to pass the verification. If the signature is indeed generated correctly, due to $s = x_{\mathrm{ID}} \cdot m \cdot T + h \cdot s_{\mathrm{ID}}$ and $s_{\mathrm{ID}} = r_{\mathrm{ID}} + h_{\mathrm{ID}} \cdot x$, we have Eq. (1). Therefore, we ensure that the signature is generated by the user with identity ID correctly if $e(s, P) = e(T, m \cdot P_{\mathrm{ID}}) \cdot e(h, R_{\mathrm{ID}}) \cdot e(h \cdot h_{\mathrm{ID}}, P_{\mathrm{pub}})$:

$$\begin{aligned} e(s, P) &= e(x_{\mathrm{ID}} \cdot m \cdot T + h \cdot s_{\mathrm{ID}}, P) \\ &= e(x_{\mathrm{ID}} \cdot m \cdot T, P) \cdot e(h \cdot (r_{\mathrm{ID}} + h_{\mathrm{ID}} \cdot x), P) \\ &= e(x_{\mathrm{ID}} \cdot m \cdot T, P) \cdot e(h \cdot r_{\mathrm{ID}}, P) \cdot e(h \cdot h_{\mathrm{ID}} \cdot x, P) \\ &= e(m \cdot T, x_{\mathrm{ID}} \cdot P) \cdot e(h, r_{\mathrm{ID}} \cdot P) \cdot e(h \cdot h_{\mathrm{ID}}, x \cdot P) \\ &= e(T, m \cdot P_{\mathrm{ID}}) \cdot e(h, R_{\mathrm{ID}}) \cdot e(h \cdot h_{\mathrm{ID}}, P_{\mathrm{pub}}) . \end{aligned} \tag{1}$$

4. Certificateless Public Verification Scheme

In this section, we will propose an efficient certificateless public auditing scheme based on the signature scheme in Sec. 3. Our scheme consists of seven algorithms: *Setup, Partial-private-extract, KeyGen, Sign, Challenge, ProofGen and ProofVerify.* The first three algorithms are same as those in Sec. 3, so we only show the details of the last four algorithms as follows.

Sign. Given a data file M with abstract index, this algorithm divides M into n blocks, $M = \{m_1, \ldots, m_n\}$. The data user with identity ID takes the system parameters $\mathrm{Param} = (G_1, G_2, q, e, P, T, H_1, H_2, P_{\mathrm{pub}})$, his own private key $\mathrm{sk}_{\mathrm{ID}} = (s_{\mathrm{ID}}, x_{\mathrm{ID}})$ and a message m_i as inputs, the generates a signature of the message. For $1 \leq i \leq n$, the user computes $h_i = H_2(m_i||\mathrm{index}||\mathrm{ID})$ and computes the signature $s_i = x_{\mathrm{ID}} \cdot m_i \cdot T + h_i \cdot s_{\mathrm{ID}}$. The user uploads the signature $S = (s_1, \ldots, s_n)$ and the message $M = (m_1, \ldots, m_n)$ to the cloud server.

Challenge. It is necessary for a verifier to check the integrity of the data in the cloud because the data owner may no longer get his file back if he did not store the data locally. The verifier randomly chooses a c-element subset I of set $[1, n]$ and a number ℓ to produce the challenge $\mathrm{chall} = \{\ell, I\}$ and send it to the cloud server.

ProofGen. After receiving the challenge $\mathrm{chall} = \{\ell, I\}$, the cloud server computes a c-element set $C = (i, v_i)$ where $i \in I$, $v_i = \ell^i \bmod q$. Based on the file $M = (m_1, \ldots, m_n)$, the cloud server computes the values $S = \sum_{i \in I} v_i \cdot s_i$, $H = \sum_{i \in I} v_i \cdot h_i$, $\mu = \sum_{i \in I} v_i \cdot m_i$ and sends proof message $\mathrm{Pro} = \{S, H, \mu\}$ to the verifier.

ProofVerify. Upon getting the proof message $\mathrm{Pro} = \{S, H, \mu\}$ from cloud server, the verifier first computes $h_{\mathrm{ID}} = H_1(\mathrm{ID}, R_{\mathrm{ID}}, P_{\mathrm{ID}})$ and then verifies the correctness of Pro by checking the equality $e(S, P) = e(T, \mu \cdot P_{\mathrm{ID}}) \cdot e(H, R_{\mathrm{ID}}) \cdot e(H \cdot h_{\mathrm{ID}}, P_{\mathrm{pub}})$. The verifier will accept the proof if the equality holds, otherwise reject it. According to the definition of public auditing,[23] the correctness means that if the file has not been destroyed, the Pro is able to pass the verification, which is

proved by checking whether the equation holds:

$$\begin{aligned}
e(S,P) &= e\left(\sum_{i\in I} v_i s_i, P\right) \\
&= e\left(\sum_{i\in I} v_i \cdot (x_{\mathrm{ID}} \cdot m_i \cdot T + h_i \cdot s_{\mathrm{ID}}), P\right) \\
&= e\left(\sum_{i\in I} v_i \cdot (x_{\mathrm{ID}} \cdot m_i \cdot T + h_i \cdot (r_{\mathrm{ID}} + h_{\mathrm{ID}} \cdot x)), P\right) \\
&= e\left(\sum_{i\in I} v_i \cdot x_{\mathrm{ID}} \cdot m_i \cdot T, P\right) \cdot e\left(\sum_{i\in I} v_i \cdot h_i \cdot r_{\mathrm{ID}}, P\right) \\
&\quad \cdot e\left(\sum_{i\in I} v_i \cdot h_i \cdot h_{\mathrm{ID}} \cdot x, P\right) \\
&= e\left(\sum_{i\in I} v_i \cdot m_i \cdot T, x_{\mathrm{ID}} \cdot P\right) \cdot e\left(\sum_{i\in I} v_i \cdot h_i, r_{\mathrm{ID}} \cdot P\right) \\
&\quad \cdot e\left(\sum_{i\in I} v_i \cdot h_i \cdot h_{\mathrm{ID}}, x \cdot P\right) \\
&= e(\mu \cdot T, P_{\mathrm{ID}}) \cdot e\left(\sum_{i\in I} v_i \cdot h_i, R_{\mathrm{ID}}\right) \cdot e\left(\sum_{i\in I} v_i \cdot h_i \cdot h_{\mathrm{ID}}, P_{\mathrm{pub}}\right) \\
&= e(T, \mu \cdot P_{\mathrm{ID}}) \cdot e(H, R_{\mathrm{ID}}) \cdot e\big(H \cdot h_{\mathrm{ID}}, P_{\mathrm{pub}}\big). \qquad (2)
\end{aligned}$$

Because $S = \sum_{i\in I} v_i \cdot s_i$, $H = \sum_{i\in I} v_i \cdot h_i$, $\mu = \sum_{i\in I} v_i \cdot m_i$ and $h_{\mathrm{ID}} = H_1(\mathrm{ID}, R_{\mathrm{ID}}, P_{\mathrm{ID}})$, we have Eq. (2). Therefore, the correctness of the proposed public verification scheme has been proved.

5. Batch Verification Scheme in the Multiuser Group

Data integrity checking is a necessary service in cloud computing. In real cloud, there is usually more than a user in a data group. Therefore, we extend our scheme to support multiuser group and make the verification scheme more efficient as we realize batch verification in this paper. Let U denote the set of users, the batch auditing scheme for multiuser is as follows. For each data user with identity ID_m, $m \in U$, the first four algorithms are same as those in Sec. 4. Then we just show the details of the other three different algorithms: **Challenge**, **ProofGen** and **ProofVerify**.

Challenge. The verifier randomly chooses a c_m-element subset I_m of set $[1, n]$ for every user in U and a number ℓ to produce the challenge $\mathrm{chall} = \{\ell, I_m\}_{m\in U}$ and send it to the cloud server.

ProofGen. After receiving the challenge $\mathrm{chall} = \{\ell, I_m\}_{m\in U}$, the cloud server computes the set $C_m = (i_m, v_{i_m})$ where $i_m \in I, v_{i_m} = \ell^{i_m} \bmod q$. Based on the file

$M_m = (m_{m_1}, \ldots, m_{m_n})$, the cloud server computes the values $S_m = \sum_{i \in I_m} v_{i_m} \cdot s_{i_m}$, $H_m = \sum_{i \in I_m} v_{i_m} \cdot h_{i_m}$, $\mu_m = \sum_{i_m \in I} v_{i_m} \cdot m_{i_m}$, then it computes $S' = \sum_{m \in |U|} S_m$, $H' = \sum_{m \in |U|} H_m$, $\mu' = \sum_{m \in |U|} \mu_m$ and sends the proof message $\text{Pro} = \{S', H', \mu'\}$ to the verifier.

ProofVerify. Upon getting the proof message $\text{Pro} = \{S', H', \mu'\}$ from cloud server, the verifier first computes $h_{\text{ID}_m} = H_1(\text{ID}_m, R_{\text{ID}_m}, P_{\text{ID}_m})$ and then checks whether does the equation $e(S', P) = e(T, \mu' \sum_{m \in |U|} P_{\text{ID}_m}) \cdot e(H', \sum_{m \in |U|} R_{\text{ID}_m}) \cdot e(H \cdot \sum_{m \in |U|} h_{\text{ID}_m}, P_{\text{pub}})$ hold to verify the correctness. The verifier will accept the proof if the elements make the equality hold, otherwise reject it.

6. Security and Performance Analysis

To measure the efficiency of our scheme, we analyze the security and performance in this section.

6.1. *Security analysis*

Theorem 1. *In the proposed scheme, no adversary $\mathcal{A}$ can forge a valid signature on a new message even if he/she has got some message-signature pairs in our scheme.*

Proof. As defined in scheme,[24] a secure CLS scheme can assist two kinds of adversaries:

- Adversary 1 as a malicious KGC: Adversary $\mathcal{A}1$ can access to the master key, but cannot replace any user's public key.
- Adversary 2 as a dishonest user: Adversary $\mathcal{A}2$ does not have access to the master key, but $\mathcal{A}2$ can replace user's public key as our scheme is certificateless.

We will prove that if $\mathcal{A}1$ or $\mathcal{A}2$ is able to forge a valid signature, then there exists an algorithm $\mathcal{F}$ that can solve the DLP. Let us consider about the issues of $\mathcal{A}1$ and $\mathcal{A}2$.

$\mathcal{A}1$: Suppose that there is a type-1 adversary $\mathcal{A}1$ for an adaptively chosen message attack against our scheme, i.e., $\mathcal{A}1$ is a (t, q_c, q_s, q_h)-forger. Then, $\mathcal{A}1$ is able to construct an algorithm $\mathcal{F}$ solving the DLP. $\mathcal{F}$ randomly picks a value $x \in Z_q$ as the system master key, sets $P_{\text{pub}} = x \cdot P$, selects an identity ID^* at random as the challenged ID and gives the public parameters $\text{Param} = (G_1, G_2, q, e, P, T, H_1, H_2, P_{\text{pub}})$ and the system master key x to $\mathcal{A}1$. Then $\mathcal{F}$ answers $\mathcal{A}1$'s queries as follows.

Create(ID): $\mathcal{A}1$ queries $\mathcal{F}$ to respond $(\text{ID}, R_{\text{ID}}, P_{\text{ID}}, s_{\text{ID}}, x_{\text{ID}}, h_{\text{ID}})$ with identity ID_i. If ID_i is in the hash list L_C which $\mathcal{F}$ maintains, $\mathcal{F}$ sends it to $\mathcal{A}1$. Otherwise, $\mathcal{F}$ chooses $a, b \in Z_q$. If $ID = ID_i$, then $\mathcal{F}$ sets $R_{\text{ID}} = a \cdot P$, $P_{\text{ID}} = Q$, $h_{\text{ID}} = H_1(\text{ID}, R_{\text{ID}}, P_{\text{ID}}) = b$, $s_{\text{ID}} = a + h_{\text{ID}} \cdot x$ and $x_{\text{ID}} = \perp$. If $\text{ID} \neq \text{ID}_i$, $\mathcal{F}$ chooses $c \in Z_q$, then sets $R_{\text{ID}} = aP, P_{\text{ID}} = bP$, $h_{\text{ID}} = H_1(\text{ID}, R_{\text{ID}}, P_{\text{ID}}) = c$, $s_{\text{ID}} = a + h_{\text{ID}} x$ and $x_{ID} = b$. Finally, $\mathcal{F}$ responds with $(\text{ID}, R_{\text{ID}}, P_{\text{ID}}, s_{\text{ID}}, x_{\text{ID}}, h_{\text{ID}})$ and inserts $(\text{ID}, R_{\text{ID}}, P_{\text{ID}}, h_{\text{ID}})$ into L_{H_1}.

H_1-query: When $\mathcal{A}1$ queries H_1 to get h_{ID} with identity ID, $\mathcal{F}$ responds with the previously defined value if the ID is already in L_{H_1}. Otherwise, $\mathcal{F}$ queries Create(ID) to get $(\mathrm{ID}, R_{\mathrm{ID}}, P_{\mathrm{ID}}, s_{\mathrm{ID}}, x_{\mathrm{ID}}, h_{\mathrm{ID}})$ and returns h_{ID}.

Partial-Private-Extract(ID): If $\mathrm{ID} = \mathrm{ID}^*$, $\mathcal{F}$ terminates. Otherwise, $\mathcal{F}$ queries Create(ID) to get $(\mathrm{ID}, R_{\mathrm{ID}}, P_{\mathrm{ID}}, s_{\mathrm{ID}}, x_{\mathrm{ID}}, h_{\mathrm{ID}})$ and returns s_{ID}.

Key-Gen(*ID*): $\mathcal{F}$ responds with the previously defined value $\mathrm{pk}_{\mathrm{ID}} = (P_{\mathrm{ID}}, R_{\mathrm{ID}})$ if the ID is already in L_C. Otherwise, $\mathcal{F}$ queries Create(ID) to get $(\mathrm{ID}, R_{\mathrm{ID}}, P_{\mathrm{ID}}, s_{\mathrm{ID}}, x_{\mathrm{ID}}, h_{\mathrm{ID}})$ and returns $\mathrm{pk}_{\mathrm{ID}}$. If $\mathrm{ID} = \mathrm{ID}^*$, $\mathcal{F}$ terminates. Otherwise, $\mathcal{F}$ responds with x_{ID}.

H_2-query: When $\mathcal{A}1$ queries H_2 to get h_{ID} with identity ID, $\mathcal{F}$ picks up $h \in Z_q$ and sets $h = H_2(m||\mathrm{index}||\mathrm{ID})$, adds $(m, \mathrm{ID}, R_{\mathrm{ID}}, P_{\mathrm{ID}}, h)$ and returns h.

Sign(ID, m): When $\mathcal{A}1$ queries on (ID, m), if $\mathrm{ID} \neq \mathrm{ID}^*$, then $\mathcal{F}$ acts as the description of our scheme. Otherwise, $\mathcal{F}$ computes $s = x_{\mathrm{ID}} \cdot m \cdot T + h \cdot s_{\mathrm{ID}} = x_{\mathrm{ID}} \cdot m \cdot T + h \cdot r_{\mathrm{ID}} + h_{\mathrm{ID}} \cdot x \cdot h$ and outputs it.

The simulation of the "Create" oracle fails if the random oracle assignment $H_1(\mathrm{ID}, R_{\mathrm{ID}}, P_{\mathrm{ID}})$ causes inconsistency. It happens with a probability equals to q_h/n. Hence, the simulation is successful q_c times with a probability $(1 - q_h/n)^{q_c}$ which is larger than $1 - \frac{q_h q_c}{n}$. The simulation of the H_2 oracle fails if the random oracle assignment $H_2(m||\mathrm{index}||\mathrm{ID})$ causes inconsistency. It happens with a probability equals to q_h/n. Hence, the simulation is successful q_h times with a probability $(1 - q_h/n)^{q_h}$ which is larger than $1 - \frac{q_h^2}{n}$. In addition, the probability of $\mathrm{ID} = \mathrm{ID}^*$ is $1/q_c$. Thus, the overall successful probability is at most $\frac{1}{q_c}(1 - \frac{q_h q_c}{n})(1 - \frac{q_h^2}{n})$.

$\mathcal{A}2$: Suppose that there is a type-2 adversary $\mathcal{A}2$ for an adaptively chosen message attack against our scheme, i.e., $\mathcal{A}2$ is a (t, q_c, q_s, q_h)-forger. Then, $\mathcal{A}2$ is able to construct an algorithm $\mathcal{F}$ solving the DLP. $\mathcal{F}$ randomly picks a value $x \in Z_q$ as the system master key, sets $P_{\mathrm{pub}} = x \cdot P$, selects an identity ID^* at random as the challenged ID and gives the public parameters $\mathrm{Param} = (G_1, G_2, q, e, P, T, H_1, H_2, P_{\mathrm{pub}})$ and the system master key x to $\mathcal{A}2$. Then $\mathcal{F}$ answers $\mathcal{A}2$'s queries as follows.

Create(ID): $\mathcal{A}2$ queries $\mathcal{F}$ to respond $(\mathrm{ID}, R_{\mathrm{ID}}, P_{\mathrm{ID}}, s_{\mathrm{ID}}, x_{\mathrm{ID}}, h_{\mathrm{ID}})$ with the identity ID_i. If ID_i is in the hash list L_C which $\mathcal{F}$ maintains, $\mathcal{F}$ sends it to $\mathcal{A}2$. Otherwise, $\mathcal{F}$ chooses $a, b, c \in Z_q$, then sets $R_{\mathrm{ID}} = a \cdot P_{\mathrm{pub}} + b \cdot P$, $P_{\mathrm{ID}} = c \cdot P$, $h_{\mathrm{ID}} = H_1(\mathrm{ID}, R_{\mathrm{ID}}, P_{\mathrm{ID}}) = a$, $s_{\mathrm{ID}} = b$ and $x_{\mathrm{ID}} = c$. Finally, $\mathcal{F}$ responds with $(\mathrm{ID}, R_{\mathrm{ID}}, P_{\mathrm{ID}}, s_{\mathrm{ID}}, x_{\mathrm{ID}}, h_{\mathrm{ID}})$ and inserts $(\mathrm{ID}, R_{\mathrm{ID}}, P_{\mathrm{ID}}, h_{\mathrm{ID}})$ into L_{H_1}. In this step, $(R_{\mathrm{ID}}, s_{\mathrm{ID}}, h_{\mathrm{ID}})$ satisfied the equation $s_{\mathrm{ID}} \cdot P = R_{\mathrm{ID}} + h_{\mathrm{ID}} \cdot P_{\mathrm{pub}}$. In other words, the secret key is valid.

H_1-query: When $\mathcal{A}2$ queries H_1 to get h_{ID} with identity ID, $\mathcal{F}$ responds with the previously defined value if the ID is already in L_{H_1}. Otherwise, $\mathcal{F}$ queries Create(ID) to get $(\mathrm{ID}, R_{\mathrm{ID}}, P_{\mathrm{ID}}, s_{\mathrm{ID}}, x_{\mathrm{ID}}, h_{\mathrm{ID}})$ and returns h_{ID}.

Partial-Private-Extract(ID): If $\mathrm{ID} = \mathrm{ID}^*$, $\mathcal{F}$ terminates. Otherwise, $\mathcal{F}$ queries Create(ID) to get $(\mathrm{ID}, R_{\mathrm{ID}}, P_{\mathrm{ID}}, s_{\mathrm{ID}}, x_{\mathrm{ID}}, h_{\mathrm{ID}})$ and returns s_{ID}.

Key-Gen(ID): $\mathcal{F}$ responds with the previously defined value $\text{pk}_{\text{ID}} = (P_{\text{ID}}, R_{\text{ID}})$ if the ID is already in L_C. Otherwise, $\mathcal{F}$ queries Create(ID) to get $(\text{ID}, R_{\text{ID}}, P_{\text{ID}}, s_{\text{ID}}, x_{\text{ID}}, h_{\text{ID}})$ and returns pk_{ID}. If $\text{ID} = \text{ID}^*$, $\mathcal{F}$ terminates. Otherwise, $\mathcal{F}$ responds with x_{ID}.

Public-Key-Replacement(ID, $\text{pk}_{\text{ID}'}$): Hash list L_R stores all tuples with the content of $(\text{ID}, r_{\text{ID}}, R_{\text{ID}}, x_{\text{ID}}, P_{\text{ID}})$ and is initialized to be empty. When $\mathcal{A}2$ queries $(\text{ID}, \text{pk}_{\text{ID}'})$, $\mathcal{F}$ sets $R_{\text{ID}} = R_{\text{ID}'}$, $P_{\text{ID}} = P_{\text{ID}'}$, $s_{\text{ID}} = \perp$ and $x_{\text{ID}} = x_{\text{ID}'}$ and adds $(\text{ID}, r_{\text{ID}'}, R_{\text{ID}'}, x_{\text{ID}'}, P_{\text{ID}'})$ to L_R.

H_2-query: When $\mathcal{A}2$ queries H_2 to get h_{ID} with identity ID, $\mathcal{F}$ picks up $h \in Z_q$ and sets $h = H_2(m||\text{index}||\text{ID})$, adds $(m, \text{ID}, R_{\text{ID}}, P_{\text{ID}}, h)$ and returns h.

Sign(ID, m): When $\mathcal{A}2$ queries on (ID, m), if $\text{ID} \neq \text{ID}^*$, $\mathcal{F}$ acts as the description of our scheme. Otherwise, $\mathcal{F}$ computes $s = x_{\text{ID}} \cdot m \cdot T + h \cdot s_{\text{ID}} = x_{\text{ID}} \cdot m \cdot T + h \cdot r_{\text{ID}} + h_{\text{ID}} \cdot x \cdot h$ and outputs it.

The simulation of the *Create* oracle fails if the random oracle assignment $H_1(\text{ID}, R_{\text{ID}}, P_{\text{ID}})$ causes inconsistency. It happens with a probability equals to q_h/n. Hence, the simulation is successful q_c times with a probability $(1 - q_h/n)^{q_c}$ which is larger than $1 - \frac{q_h q_c}{n}$. The simulation of the H_2 oracle fails if the random oracle assignment $H_2(m||\text{index}||\text{ID})$ causes inconsistency. It happens with a probability equals to q_h/n. Hence, the simulation is successful q_h times with a probability $(1 - q_h/n)^{q_h}$ which is larger than $1 - \frac{q_h^2}{n}$. In addition, the probability of $\text{ID} = \text{ID}^*$ is $1/q_c$. Thus, the overall successful probability is less than $\frac{1}{q_c}(1 - \frac{q_h q_c}{n})(1 - \frac{q_h^2}{n})$.

As discussed above, if $\mathcal{A}1$ or $\mathcal{A}2$ is able to successfully generate a forgery of a signature, then $\mathcal{F}$ is able to solve the DL problem, which contradicts to the DLA. Therefore, it is hard to generate a forgery of a signature. □

Theorem 2. *No replace attack, which means the cloud server is able to pass the checking with other message rather than the challenged message, exists in our auditing protocol.*

Proof. The cloud server can pass the verification when and only when all the challenged data blocks and their signatures are correct. If any one of them is corrupted or not up-to-date on the server, the server cannot pass the checking since the equation does not hold. If the cloud executes a replace attack to pass the verification, he needs to use a wrong set such as $\{s_l, h_l, m_l\}$ to replace $\{s_j, h_j, m_j\}$. And he generates the proof message as $S' = v_j \cdot s_l + \sum_{i \in I, i \neq j} v_i \cdot s_i$, $H' = v_j h_l + \sum_{i \in I, i \neq j} v_i \cdot h_i$, $\mu' = v_j \cdot m_l + \sum_{i \in I, i \neq j} v_i \cdot m_i$. Then, we have the relation $e(S', P) = e(v_j \cdot (h_l - h_j) \cdot h_{\text{ID}} + \sum_{i \in I} v_i \cdot h_i \cdot h_{\text{ID}}, P_{\text{pub}})$ which can be proven as shown in Eq. (3).

When and only when $h_l - h_j = 0$, the cloud server is able to pass the verification. The fact is that due to the collision resistance of hash functions, $h_l - h_j$ cannot be equal to zero. In other words, the cloud server cannot pass the auditing in our scheme. Hence,

our scheme can resist the replace attack:

$$
\begin{aligned}
e(S', P) &= e(v_j \cdot s_l + \sum_{i \in I, i \neq j} v_i \cdot s_i, P) \\
&= e(v_j \cdot (x_{\mathrm{ID}} \cdot m_l \cdot T + h_j \cdot s_{\mathrm{ID}}) + \sum_{i \in I, i \neq j} v_i \cdot (x_{\mathrm{ID}} \cdot m_i \cdot T + h_i \cdot s_{\mathrm{ID}}), P) \\
&= e(v_j \cdot (x_{\mathrm{ID}} \cdot m_l \cdot T + h_j \cdot (r_{\mathrm{ID}} + h_{\mathrm{ID}} \cdot x)) \\
&\quad + \sum_{i \in I, i \neq j} v_i \cdot (x_{\mathrm{ID}} \cdot m_i \cdot T + h_i \cdot (r_{\mathrm{ID}} + h_{\mathrm{ID}} \cdot x)), P) \\
&= e(v_j \cdot x_{\mathrm{ID}} \cdot m_l \cdot T + \sum_{i \in I, i \neq j} v_i \cdot x_{\mathrm{ID}} \cdot m_i \cdot T, P) \cdot e(v_j \cdot h_l \cdot r_{\mathrm{ID}} \\
&\quad + \sum_{i \in I, i \neq j} v_i \cdot h_i \cdot r_{\mathrm{ID}}, P) \cdot e(v_j \cdot h_l \cdot h_{\mathrm{ID}} \cdot x + \sum_{i \in I, i \neq j} v_i \cdot h_i \cdot h_{\mathrm{ID}} \cdot x, P) \\
&= e(v_j \cdot m_l \cdot T + \sum_{i \in I, i \neq j} v_i \cdot m_i \cdot T, x_{\mathrm{ID}} \cdot P) \cdot e(v_j \cdot h_l + \sum_{i \in I, i \neq j} v_i \cdot h_i, \\
&\quad r_{\mathrm{ID}} \cdot P) \cdot e(v_j \cdot h_l \cdot h_{\mathrm{ID}} + \sum_{i \in I, i \neq j} v_i \cdot h_i \cdot h_{\mathrm{ID}} \cdot x, P) \\
&= e(v_j \cdot m_l \cdot T + \sum_{i \in I, i \neq j} v_i \cdot m_i \cdot T, P_{\mathrm{ID}}) \cdot e(v_j \cdot h_l + \sum_{i \in I, i \neq j} v_i \cdot h_i, R_{\mathrm{ID}}) \\
&\quad \cdot e(v_j \cdot h_l \cdot h_{\mathrm{ID}} + \sum_{i \in I, i \neq j} v_i \cdot h_i \cdot h_{\mathrm{ID}}, P_{\mathrm{pub}}) \\
&= e(T, \mu' \cdot P_{\mathrm{ID}}) \cdot e(v_j \cdot (h_l - h_j) + \sum_{i \in I} v_i \cdot h_i, R_{\mathrm{ID}}) \\
&\quad \cdot e(v_j \cdot (h_l - h_j) \cdot h_{\mathrm{ID}} + \sum_{i \in I} v_i \cdot h_i \cdot h_{\mathrm{ID}}, P_{\mathrm{pub}}) \,. \qquad (3)
\end{aligned}
$$

□

Theorem 3. *The public verifier cannot get any information about the outsourced data from the values he received during the verification.*

Proof. In the verify procedure, a public verifier can obtain $\mathrm{chall} = \{\ell, I\}$ and $\mathrm{Pro} = (S, H, \mu)$. Then, the verifier is able to get the information from Pro. It is clear that the information of data owner cannot be recovered from Pro as these values are with the following form: $S = \sum_{i \in I} v_i \cdot s_i$, $H = \sum_{i \in I} v_i \cdot h_i$ and $\mu = \sum_{i \in I} v_i \cdot m_i$.

Although the verifier can compute all $i \in I, v_i = \ell^i \bmod q$, he cannot obtain all s_i, h_i, m_i from the equations above. However, they are linear with unknown $|I|$ variables, respectively, where $|I|$ may be 460. It means that they have $q^{|I|}$ solutions to test the equation $S = \sum_{i \in I} v_i \cdot s_i$, $H = \sum_{i \in I} v_i \cdot h_i$ and $\mu = \sum_{i \in I} v_i \cdot m_i$ and only one of them is the real value the verifier wants to get. In other words, the probability of obtaining right s_i, h_i, m_i is $q^{-|I|}$. Furthermore, it is impossible for the verifier to extract m_i from h_i since hash function is a one-way function. Therefore, the verifier does not have the ability to obtain any information about the data block during the verification process. □

6.2. *Performance analysis*

We first evaluate the performance of our scheme by calculating the communication and calculation overheads. Then we compare the computation overhead of our scheme with schemes CPA[23] and CLPA.[25] The notations of operations throughout this paper are listed in Table 1.

6.2.1. *Computation cost*

In *ProofGen*, a public verifier sends audit challenge chall $= \{\ell, I\}$ to the cloud. Then the cloud generates the audit proof message $\text{Pro} = (S, H, \mu)$ and returns it to the verifier. The computation overhead of generating audit proof is $c\text{Exp}_{G_1} + 3c\text{Mul}_{G_1}$ and that of verifying a proof is $4\text{Pair} + 4\text{Mul}_{G_1} + \text{Hash}_{G_1}$. We compare them with the schemes CPA[23] and CLPA[25] in Table 2.

When we audit several files together, batch auditing is more efficient than normal auditing. To show the difference between normal auditing and batch auditing, we list the computation overheads of them in Table 3. Note that we use m to denote the number of auditing files.

Table 1. Notations of cryptographic operations.

Symbol	Meanings
$\text{Exp}_{G_1}, \text{Exp}_{G_2}$	Exponentiations in G_1, G_2
$\text{Mul}_{G_1}, \text{Mul}_{G_2}$	Multiplications in G_1, G_2
Hash_{G_1}	Hash operation in G_1
Pair	Pair operation
$\lvert A \rvert$	Length of A

Table 2. Comparison of public auditing computation costs.

	CPA	CLPA	Proposed
ProofGen	$c\text{Exp}_{G_1} + c\text{Mul}_{G_1}$	$c\text{Exp}_{G_1} + c\text{Mul}_{G_1}$	$c\text{Exp}_{G_1} + 3c\text{Mul}_{G_1}$
ProofVerify	$3\text{Pair} + (3c + k) \times (\text{Exp}_{G_1} + \text{Mul}_{G_1}) + \text{Mul}_{G_2} + c\text{Hash}_{G_1}$	$2\text{Pair} + (c+3)\text{Exp}_{G_1} + (c+2)\text{Mul}_{G_1} + (c+1)\text{Hash}_{G_1} + 2\text{Hash}_{Z_p}$	$4\text{Pair} + 4\text{Mul}_{G_1} + \text{Hash}_{G_1}$
Total	$3\text{Pair} + (2c + k)(\text{Exp}_{G_1} + \text{Mul}_{G_1}) + \text{Mul}_{G_2} + c\text{Hash}_{G_1}$	$2\text{Pair} + (2c+3)\text{Exp}_{G_1} + (2c+2)\text{Mul}_{G_1} + (c+1)\text{Hash}_{G_1} + 2\text{Hash}_{Z_p}$	$4\text{Pair} + c\text{Exp}_{G_1} + (3c+4)\text{Mul}_{G_1} + \text{Hash}_{G_1}$

Note: The meanings of notations are in Table 1.

Table 3. Comparison of normal auditing and batch auditing.

	Computation overhead
Normal auditing	$m[4\text{Pair} + c\text{Exp}_{G_1} + (3c+4)\text{Mul}_{G_1} + \text{Hash}_{G_1}]$
Batch auditing	$4\text{Pair} + mc\text{Exp}_{G_1} + (3mc+4)\text{Mul}_{G_1} + m\text{Hash}_{G_1}$

6.2.2. *Communication cost*

To check the integrity of data in the cloud, a public verifier sends an auditing challenge $\text{chall} = \{\ell, I\}$ to the cloud first, and then the cloud needs to send an auditing response $\text{Por} = (S, H, \mu)$ back to the public verifier. In **ProofGen** and **ProofVerify**, similar to CPA[23] and CLPA,[25] the communication cost of an auditing challenge is $|q| + c|n|$ bits, and the communication cost of an auditing response is $3|q|$ bits, where n is the total number of blocks, $|q|$ is the length of an element of Z_q, and $|p|$ is the length of an element of Z_p.

6.2.3. *Experimental results*

In the following experiments, we utilize the pairing-based cryptography (PBC)[26] library to simulate the cryptographic operations in our scheme, and all the experiments are tested on Ubuntu system with Intel Core i7 3.40 GHz processor over 1,000 times. We assume $|q| = 160$ bits, $|p| = 160$ bits and the number of blocks in shared data is $n = 1,000,000$ and $|n| = 20$ bits. Every block has k elements where $k = 1$ as we did not split m_i as in CPA.[23] According to previous work,[4] to keep the detection probability greater than 99%, we set the number of selected blocks in an auditing task as $c = 460$.

Table 4 and Fig. 2 show the comparison of experimental results between our scheme and the schemes CPA[23] and CLPA.[25] It is obvious that our scheme is more efficient. The communication cost of our scheme is the same as that of schemes CPA[23] and CLPA.[25] The lower the communication cost, the better it is for mobile storage.

To show the efficiency of batch auditing, we show the difference between normal auditing and batch auditing in Fig. 3. Batch auditing means that several files can be verified together and it needs less auditing time than normal auditing which has to audit these files one by one. As shown in Table 3, we audit c blocks of a file and the number of files is m. In Fig. 3, we set c as 460 to keep the detection probability greater than 99%. It is easy to find that the overhead of normal auditing is same as in batch auditing when m is equal to 1. As batch auditing aggregates some elements, it costs less than normal auditing which needs more "Pair" and "Hash" operations.

Table 4. Experimental results comparison (ms).

	CPA	CLPA	Our work
	$5.951c$+$0.016k$+9.357	$5.935c$+12.17	$0.03c$+18.379
$c = 300$	1,795	1,793	27
$c = 460$	2,747	2,742	32

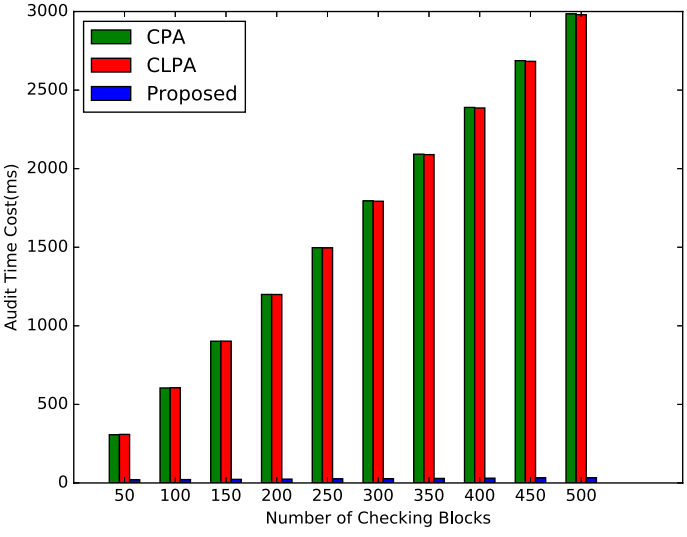

Fig. 2. Experimental results comparison (ms) ($k = 1$).

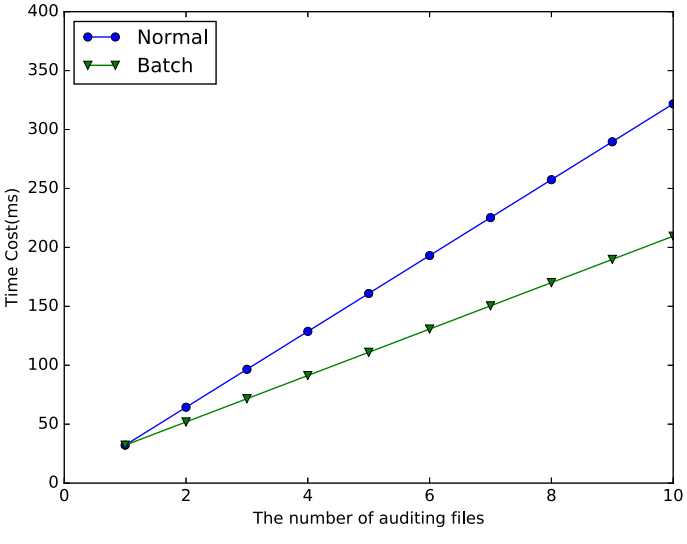

Fig. 3. Experimental results of normal auditing and batch auditing (ms)($c = 460$).

7. Conclusion

In this paper, we propose an efficient certificateless signature with the knowledge of elliptic curve and bilinear map. Then we design a certificateless public verification mechanism based on the signature to check the integrity of data outsourced in cloud.

Specifically, the scheme allows a public verifier who does not need to manage certificates during the verification other than the data owner to check for data integrity. Meanwhile, a verifier checks the integrity without retrieving the entire data. To consider the real cloud storage environment, we further extend it to support batch verification which is more efficient in multiuser groups than in other public auditing schemes. We prove that our scheme is provably secure under the discrete logarithm assumption and the experimental results shows that our scheme is more efficient.

Acknowledgments

The authors are very grateful to the editor and reviewers for their suggestions to improve the quality of paper. This work was supported by the National Natural Science Foundation of China (Grant Nos. 61272420 and 61502234) and the Natural Science Foundation of Jiangsu Province of China (Grant No. BK20150785).

References

1. Y. Chen, F. Li, B. Du, J. Fan and Z. Deng, A quantitative analysis on semantic relations of data blocks in storage systems, *J. Circuits Syst. Comput.* **24** (2015) 1550118.
2. S. Yu, Challenges and opportunities of privacy study in the age of big data, *IEEE Access* **4** (2016) 2751–2763.
3. W. Ren, L. Zeng, R. Liu and C. Cheng, F2AC: A lightweight, fine-grained, and flexible access control scheme for file storage in mobile cloud computing, *Mobile Inf. Syst.* **2016** (2016) 5232846.
4. A. Fu, N. Qin, J. Song and M. Su, Privacy-preserving public auditing for multiple managers shared data in the cloud, *J. Comput. Res. Dev.* **52** (2015) 2353–2362.
5. G. Xie, G. Zeng, Y. Chen, Y. Bai, Z. Zhou, R. Li and K. Li, Minimizing redundancy to satisfy reliability requirement for a parallel application on heterogeneous service-oriented systems, *IEEE Trans. Serv. Comput.* **PP** (2017) 1–1.
6. G. Ateniese, R. Burns and R. Curtmola, Provable data possession at untrusted stores, *Proc. 14th ACM Conf. Computer and Communications Security* (Springer, Berlin, 2007), pp. 598–609.
7. G. Ateniese, R. Burns and R. Curtmola, Remote data checking using provable data possession, *ACM Trans. Inf. Syst. Secur.* **14** (2011) 12.
8. A. Juels and B. S. Kaliski, PORs: Proofs of retrievability for large files, *Proc. 14th ACM Conf. Computer and Communications Security* (Springer, Berlin, 2007), pp. 584–597.
9. M. Long, Y. Li and F. Peng, Integrity verification for multiple data copies in cloud storage based on spatiotemporal chaos, *Int. J. Bifurcation Chaos Appl. Sci. Eng.* **27** (2017) 1750054.
10. Y. Zhu, S. Wang and H. Hu, Secure collaborative integrity verification for hybrid cloud environments, *Int. J. Coop. Inf. Syst.* **21** (2012) 165–197.
11. B. Wang, B. Li and H. Li, Oruta: Privacy-preserving public auditing for shared data in the cloud, *IEEE Trans. Cloud Comput.* **2** (2014) 43–56.
12. L. Huang, G. Zhang and A. Fu, Privacy-preserving public auditing for dynamic group based on hierarchical tree, *J. Comput. Res. Dev.* **53** (2016) 2334–2342.
13. L. Huang, G. Zhang and A. Fu, Privacy-preserving public auditing for non-manager group, *Proc. 2017 IEEE Int. Conf. Communications (ICC)* (2017).

14. C. Wang, S. S. M. Chow, Q. Wang, K. Ren and W. Lou, Privacy-preserving public auditing for secure cloud storage, *IEEE Trans. Comput.* **62** (2012) 362–375.
15. L. Huang, G. Zhang and A. Fu, Certificateless public verification scheme with privacy-preserving and message recovery for dynamic group, *Proc. ACM Australasian Computer Science Week Multiconf.* (2017).
16. Y. Yu, M. H. Au and Y. Mu, Enhanced privacy of a remote data integrity-checking protocol for secure cloud storage, *Int. J. Inf. Sec.* **14** (2015) 307–318.
17. M. S. Hwang, T. H. Sun and C. C. Lee, Achieving dynamic data guarantee and data confidentiality of public auditing in cloud storage service, *J. Circuits Syst. Comput.* **26** (2016) 1750072.
18. Y. Yu, L. Xue, H. A. Man, W. Susilo, J. Ni, Y. Zhang, A. V. Vasilakosc and J. Shen, Cloud data integrity checking with an identity-based auditing mechanism from RSA, *Future Gener. Comput. Syst.* **62** (2016) 85–91.
19. H. Wang, Identity-based distributed provable data possession in multicloud storage, *IEEE Trans. Serv. Comput.* **8** (2015) 328–340.
20. M. S. Kiraz, İ. Sertkaya and O. Uzunkol, An efficient ID-based message recoverable privacy-preserving auditing scheme, *Proc. 2015 13th Annu. Conf. Privacy, Security and Trust (PST)* (2015), pp. 117–124.
21. J. Zhang and Q. Dong, Efficient ID-based public auditing for the outsourced data in cloud storage, *Inf. Sci.* **334–344** (2016) 1–14.
22. S. Tang and G. Wei, ID-based digital multisignature scheme, *J. Circuits Syst. Comput.* **9** (1999) 223–227.
23. B. Wang, B. Li, H. Li and F. Li, Certificateless public auditing for data integrity in the cloud, *Proc. 2013 IEEE Conf. Communications and Network Security (CNS)* (2013), pp. 136–144.
24. S. S. Al-Riyami and K. G. Paterson, Certificateless public key cryptography, *ASIACRYPT 2003: Advances in Cryptology*, Lecture Notes in Computer Science, Vol. 2894 (Springer, Berlin, 2003), pp. 452–473.
25. D. He, S. Zeadally and L. Wu, Certificateless public auditing scheme for cloud-assisted wireless body area networks, *IEEE Syst. J.* **PP** (2015) 1–10.
26. B. Lynn, PBC: The pairing-based cryptography library (2011), http://crypto. stanford. edu/pbc.