JOURNAL OF INFORMATION SCIENCE AND ENGINEERING 35, 635-650 (2019)
DOI: 10.6688/JISE.201905_35(3).0009

Secure Outsourcing Algorithm for Signature Generation in Privacy-Preserving Public Cloud Storage Auditing

PU ZHAO[1], JIA YU[1,2,+] AND HANLIN ZHANG[1]
[1]*College of Computer Science and Technology*
Qingdao University
Qingdao, 266071 P.R. China
[2]*State Key Laboratory of Information Security*
Institute of Information Engineering
Chinese Academy of Sciences
Beijing, 100093 P.R. China
E-mail: zhaopu789@126.com; qduyujia@163.com; hanlin10001@icloud.com

Public cloud storage auditing allows a file owner or a public verifier to conduct integrity checking without downloading the whole file from cloud server. Plenty of unaffordable modular exponentiations for resource-constraint devices are required on the client side in the process of signature generation for public cloud storage auditing. In this paper, we introduce a secure outsourcing algorithm to make cloud storage public auditing more feasible and practical for the computationally limited clients. Compared with the existing algorithms for secure outsourcing of modular exponentiations, the proposed algorithm designed for this scenario takes advantage of that the bases of modular exponentiations do not need to be protected. Therefore, our algorithm is efficient. Furthermore, the evaluation shows that it has high efficiency and checkability as well. Specifically, the client-side cost is reduced to 5.6% of the original computation cost, and the probability that the client detects the misbehavior of server is close to 1.

Keywords: public auditing, cloud storage, secure outsourcing, modular exponentiation, cloud computing

1. INTRODUCTION

With the development of information technology, cloud services are widely used because it has a lot of advantages. More and more people choose to store data on cloud such as iCloud, Amazon Web Services and OneDrive to save storage cost. Although cloud storage's benefits are superb, security and privacy concerns are the primary obstacles to its wide adoption [1]. The data stored on the cloud be lost or modified by someone malicious including the cloud service providers themselves. Therefore, the integrity of outsourced files needs to be verified.

In recent years, the concept of public auditing has been presented, which allows a third party to perform integrity checking without acquiring the entire data on the cloud. In public auditing, the file is divided into many small blocks, and verifier does integrity checking based on some random blocks instead of the entire data. However, in most of public auditing schemes, the data owner's privacy might be violated. In other words, it is possible that auditors collect some confidential information about outsourced data. In order to solve this problem, Wang *et al*. [2] presented an improved auditing scheme

Received November 1, 2017; revised November 11, 2018; accepted January 4, 2019.
Communicated by Jiann-Liang Chen.
[+] Corresponding author.

named WWRL, so that public verifiers are unable to get anything private about users' private data. However, WWRL cannot protect users' identity privacy. To protect users' identity privacy from public verifiers, Wang *et al*. [3] proposed a novel privacy-preserving public auditing mechanism named Oruta. Ring signatures are utilized in Oruta to construct homomorphic authenticators, so that the identity of the signer could be kept private from the public verifier. However, a deficiency of Oruta is the computation-intensive task. Oruta needs many modular exponentiations which are relatively expensive for resource-constrained devices such as mobile terminals. Therefore, we try to make Oruta more practical by secure outsourcing computation in this paper.

1.1 Our Contributions

In this paper, we propose an outsourcing algorithm **RS** to outsource modular exponentiations in signature generation to a single (untrusted) computation server. We divide the ring signature generation into 3 sub-problems and outsource them respectively. Our algorithm takes advantage of the fact that the bases of modular exponentiations in the first 2 sub-problems do not need to be protected. By taking advantage of this feature, we greatly improve the efficiency of our algorithm. Besides, the proposed algorithm provides good input/output privacy and checkability. Specific analysis is given after algorithm description.

1.2 Related Work

Public Auditing. Ateniese *et al*. [4] first introduced the model for PDP that allows a client who has stored data at an untrusted server to check the integrity without downloading the original data. Wang *et al*. [2] proposed a modified auditing mechanism named WWRL. WWRL protects the content of users' private data from being disclosed to any public verifiers. Based on ring signature [5], Wang *et al*. [3] presented an advanced privacy-preserving public auditing mechanism which is called Oruta. Compared with old schemes, Oruta preserves identity privacy of the signer on each block in shared data through ring signatures. Proofs of retrievability (PoR) was first proposed by Juels and Kaliski [6], which enables the cloud storage server to convince the clients that the remotely stored data can be entirely retrieved. This method based on sentinel blocks which are hidden among other data blocks before remote storage. Shacham and Waters [7] introduced verifiable PoR schemes, which provide private and public verifiability respectively against the strongest adversary. Yu *et al*. [8, 9] focused on how to mitigate the damage of the client's key exposure in cloud storage auditing, and proposed a new solution by using binary-tree based key updates technique [10]. Zhang *et al*. [11] proposed a storage auditing scheme achieving highly-efficient user revocation. Shen *et al*. proposed a novel storage auditing scheme with sensitive information hiding in [12].

Secure Outsourcing. In 1992, Chaum and Pedersen [13] proposed the notion of wallets with observers, which were regarded as an embryonic form of secure outsourcing computation. In 2005, Hohenberger and Lysyanskaya [14] presented the formal security definition of outsourcing computation. In [14], Hohenberger *et al*. also proposed an algorithm for outsourcing modular exponentiations to two untrusted servers. However, the

checkability of the results in this scheme is only $\frac{1}{2}$. Chen *et al*. [15] improved the checkability to $\frac{2}{3}$. Wang *et al*. [16] first presented a secure outsourcing scheme that supports variable-exponent variable-based multi-exponentiations under the single untrusted server model. Based on [16], clients are able to speed up PDP schemes. Ding *et al*. [17] presented an algorithm for outsourcing modular exponentiations under single untrusted program model, whose checkability was close to 1. Chen *et al*. [18] proposed an algorithm for outsourcing large-scale linear equations, whose basic idea is blinding coefficient matrix by sparse matrix. The efficiency and checkability of algorithm in [18] are superb. Tian *et al*. [19] introduced an algorithm that makes bilinear pairing computing more efficient and flexible. Lei *et al*. [20] found a method of how to outsource determinant computation to a remote cloud server. Yu *et al*. [21] proposed verifiable key updates outsourcing technique for cloud storage auditing.

2. SYSTEM MODEL AND DEFINITIONS

2.1 System Model

As shown in Fig. 1, an outsourcing computation architecture is described as follows: The resource-constrained client C wants to complete a heavy computation task $T : D \rightarrow M$ which is hard to be accomplished by C locally. Therefore, C asks for help from a cloud computation server S. However, S may be a malicious server. It means that S may either steal C's private information, or return an invalid computation result to cheat C, or even both. S is not fully trusted by C, so C first encrypts $F \in D$ to an encrypted version. Then, C submits the encrypted F to S, and asks S to find the computation result about the encrypted F. S performs the computation task and returns the computation result about the encrypted F. After receiving this answer, C checks the correctness of it. If it is correct, C decrypts it to get the original answer $R \in M$ about F. In the whole process, the cloud server should not learn anything sensitive about F and the answer R.

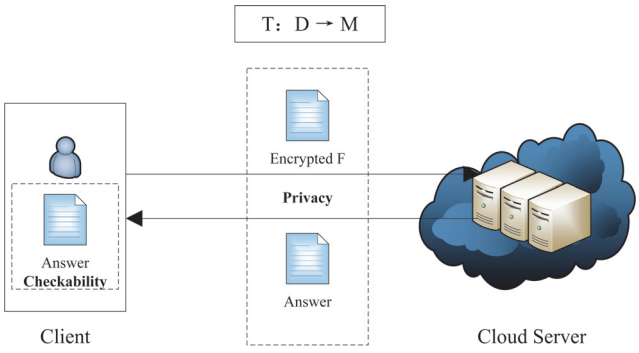


Fig. 1. Outsourcing computation architecture.

2.2 Formal Definition

Gennaro *et al*. [22] proposed a formal definition for secure outsourcing computation.

Definition 1: A secure outsourcing computation scheme consists of 5 parts:

OCS = (KeyGen, ProbGen, Compute, Verify, Solve).

These five algorithms are defined as follows:

1. KeyGen(λ, F) $\rightarrow$ (PK, SK): C invokes the randomized key generation algorithm to generate a public key PK to encode the target function F according to security parameter λ. At the same time, this algorithm also produces a corresponding secret key SK which is kept secret by the client C.
2. ProbGen$_{SK}(x) \rightarrow (\sigma_x, \tau_x)$: C invokes the problem generation algorithm to encode the function input x as a public value σ_x which is submitted to server S. Meanwhile, this algorithm generates a secret value τ_x which is kept private by the client C.
3. Compute$_{PK}(\sigma_x) \rightarrow \sigma_y$: On the basis of PK and encrypted value σ_x, server S computes an encoded version σ_y of the function's output $y = F(x)$.
4. Verify$_{SK}(\sigma_y) \rightarrow 1 \cup 0$: C checks the correctness of σ_y according to secret key SK. If σ_y is valid, the verification algorithm outputs 1. Otherwise it outputs 0.
5. Solve$_{SK}(\tau_x, \sigma_y) \rightarrow y$: C invokes the solving algorithm and takes the secret key SK, secret value τ_x and the encoded answer σ_y as input, then C gets the original result $y = F(x)$.

2.3 Security Requirements

Next, we introduce some security requirements for outsourcing computation.

The first requirement is the privacy for the input/output of the whole algorithm. In other words, the server S cannot learn anything sensitive about the input/output from its communication with client C.

Definition 2 (Privacy [22]): Given a security parameter λ, a pair of algorithms (C, S) is said to be input/output privacy if for any probabilistic polynomial time adversary $\mathscr{A}$,

$$\mathrm{Adv}_{\mathscr{A}}^{CS}(F, \lambda) \leq \mathrm{negli}(\lambda),$$

where negli() is a negligible function of its input.

$$\mathrm{Adv}_{\mathscr{A}}^{CS}(F, \lambda) = \left| \mathrm{Prob}[b = b'] - \frac{1}{2} \right|$$

is defined as the advantage of $\mathscr{A}$ in the experiment as follows:

$$(PK, SK) \xleftarrow{R} \mathrm{KeyGen}(\lambda, F);$$
$$(x_0, x_1) \leftarrow \mathscr{A}^{\mathrm{PubProbGen}_{SK}(\cdot)}(PK)$$
$$(\sigma_0, \tau_0) \leftarrow \mathrm{ProbGen}_{SK}(x_0);$$
$$(\sigma_1, \tau_1) \leftarrow \mathrm{ProbGen}_{SK}(x_1);$$
$$b \leftarrow \{0,1\};$$
$$b' \leftarrow \mathscr{A}^{\mathrm{PubProbGen}_{SK}(\cdot)}(PK, x_0, x_1, \sigma_b)$$

During this experiment, the adversary $\mathscr{A}$ could call for the encoding of any input he wants. $\text{PubProbGen}_{SK}(x)$ invokes $\text{ProbGen}_{SK}(x)$ to generate (σ_x, τ_x), and returns only the public part σ_x. In a word, $\text{PubProbGen}_{SK}(x)$ is probabilistic.

The second requirement is the efficiency of the outsourcing algorithms. Intuitively, the calculation done by the client C in outsourcing algorithms should be significantly less than that to complete the computation task by itself.

Definition 3 (α-Efficiency [14]): A pair of algorithms (C, S) is said to be an α-efficient implementation of an algorithm F if (1) C^S correctly implements F and (2) for $\forall x$, the running time of C^S is less than or equal to an α-multiplicative factor of the running time of F.

The last requirement is checkability. The invalid output produced by malicious server should not pass the validation and the client C could detect the inaccuracy with high probability.

Definition 4 (β-checkable [14]): A pair of algorithms (C, S) is said to be a β-checkable implementation of an algorithm F if (1) C^S correctly implements F and (2) for $\forall x$, if a malicious server S' deviates from its presupposed functionality during the execution of $C^S(x)$, C will detect the inaccuracy with probability no less than β.

3. OUTSOURCING ALGORITHM FOR SIGNATURE GENERATION

In this section, firstly we review the signature generation in Oruta [3]. Then we propose our outsourcing algorithm **RS**.

3.1 Signature Generation

There are three multiplicative cyclic groups of order p: $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$. Let g_1 and g_2 be generators of $\mathbb{G}_1$ and $\mathbb{G}_2$ respectively. Let $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ be a bilinear map and $\psi : \mathbb{G}_2 \to \mathbb{G}_1$ be a computable isomorphism with $\psi(g_2) = g_1$. There are three hash functions $H_1 : \{0, 1\}^* \to \mathbb{G}_1$, $H_2 : \{0, 1\}^* \to \mathbb{Z}_q$ and $h : \mathbb{G}_1 \to \mathbb{Z}_p$. The number of users in the group is d. Shared data M is divided into n blocks as $M = (m_1, m_2, ..., m_n)$. Furthermore, each block $m_j = (m_{j,1}, m_{j,2}, ..., m_{j,k})$ is divided into k elements of $\mathbb{Z}_p$, and these k elements are also called k sections. $(e, \psi, p, q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, H_1, H_2, h, d, n, k)$ are public parameters.

User u_i picks $x_i \in \mathbb{Z}_p$, and computes $w_i = g_2^{x_i}$. User u_i's public key is $\text{pk}_i = w_i$, and his or her secret key is $\text{sk}_i = x_i$. The original user also randomly picks a public aggregate key $\text{pak} = (\eta_1, \eta_2, ..., \eta_k)$, where η_l are random elements of $\mathbb{G}_1$.

Algorithm 1: Key-Generation Algorithm

Input: A security parameter κ.

Output: Outsourcing secret key K: $r_1, r_2, r_3, \{t_1, t_1', t_2, t_2', z_1, z_2\} \in \mathbb{Z}_p$, $a_{j,i} \in \mathbb{Z}_p$, $(b_1, g_1^{b_1})$, $(b_2, g_1^{b_2})$, $(b_3, g_1^{b_3})$, $(b_4, g_1^{b_4})$, $(b_5, g_1^{b_5})$ and $(b_6, g_1^{b_6})$.

1: C randomly chooses three small integers r_1, r_2 and r_3 greater than 1 in a small rage.

2: On input a security parameter κ, C generates a series of integers randomly including

$\{t_1, t'_1, t_2, t'_2, z_1, z_2\}$ which are all belong to $\mathbb{Z}_p$.
3: Client C chooses $a_{j,i} \in \mathbb{Z}_p$ randomly.
4: C generates 6 blinding pairs $(b_1, g_1^{b_1})$, $(b_2, g_1^{b_2})$, $(b_3, g_1^{b_3})$, $(b_4, g_1^{b_4})$, $(b_5, g_1^{b_5})$ and $(b_6, g_1^{b_6})$.

According to block $m_j = (m_{j,1}, m_{j,2}, ..., m_{j,k})$, this block's identifier id_j, public keys $(pk_1, pk_2, ..., pk_d) = (w_1, w_2, ..., w_d)$ and the secret key sk_s, user u_s computes a signature for this block as follows:

1. User u_s aggregates block m_j through the public aggregate key pak, and calculates

$$\beta_j = H_1(id_j) \prod_{l=1}^{k} \eta_l^{m_{j,l}} \in \mathbb{G}_1.$$

2. User u_s chooses $a_{j,i} \in \mathbb{Z}_p$ randomly, and computes $\sigma_{j,i} = g_1^{a_{j,i}}$ for all $i \neq s$. Then, user u_s computes

$$\sigma_{j,s} = \left(\frac{\beta_j}{\psi(\prod_{i \neq s} w_i^{a_{j,i}})} \right)^{1/x_s}.$$

The signature of block m_j is $\sigma_j = (\sigma_{j,1}, ..., \sigma_{j,d})$. Actually, it is a ring signature for this block.

3.2 Outsourcing Algorithm

Algorithm 2: Algorithm of computing β_j

Input: K, pak $= (\eta_1, \eta_2, ..., \eta_k)$, $m_j = (m_{j,1}, m_{j,2}, ..., m_{j,k})$ and its identifier id_j.
Output: "error" or β_j.

1: Let $mul_j = \prod_{l=1}^{k} \eta_l^{m_{j,l}}$, C computes u_l, u'_l, θ_1 and θ'_1 which satisfy $m_{j,l} = u_l + t_1\theta_1$ and $r_1 m_{j,l} = u'_l + t'_1\theta'_1$, where $l = 1, 2, ..., k$. After that, mul_j and $mul_j^{r_1}$ are divided as:

$$mul_j = \prod_{l=1}^{k} \eta_l^{m_{j,l}} = (\prod_{l=1}^{k} \eta_l^{u_l}) \cdot ((\eta_1\eta_2...\eta_k)^{\theta_1})^{t_1},$$

$$mul_j^{r_1} = \prod_{l=1}^{k} \eta_l^{r_1 m_{j,l}} = (\prod_{l=1}^{k} \eta_l^{u'_l}) \cdot ((\eta_1\eta_2...\eta_k)^{\theta'_1})^{t'_1}.$$

2: Client C queries server S in random order as:

$$S(\eta_l, u_l) \to \eta_l^{u_l};$$
$$S(\eta_l, u'_l) \to \eta_l^{u'_l};$$
$$S((\eta_1\eta_2...\eta_k), \theta_1) \to (\eta_1\eta_2...\eta_k)^{\theta_1};$$
$$S((\eta_1\eta_2...\eta_k), \theta'_1) \to (\eta_1\eta_2...\eta_k)^{\theta'_1},$$

where $l = 1, 2, ..., k$.

3: Server S computes them and returns the answer to Client C.
4: Client C receives the answer from server S and checks the outputs. *i.e.*, client C checks whether the equation holds:

$$\left((\prod_{l=1}^{k} \eta_l^{u_l}) \cdot ((\eta_1 \eta_2 ... \eta_k)^{\theta_1})^{t_1} \right)^{r_1} = (\prod_{l=1}^{k} \eta_l^{u'_l}) \cdot ((\eta_1 \eta_2 ... \eta_k)^{\theta'_1})^{t'_1}.$$

If not, C outputs "error". Otherwise, C computes

$$\beta_j = H_1(id_j)(\prod_{l=1}^{k} \eta_l^{u_l}) \cdot ((\eta_1 \eta_2 ... \eta_k)^{\theta_1})^{t_1}.$$

The proposed algorithm **RS** starts by invoking Algorithm 1 (Key-Generation) to set up an outsourcing secret key. For the purpose of distinguishing between secret key used in outsourcing and secret key used in signature generation, we call the former as outsourcing secret key.

After invoking Algorithm 1, C obtains outsourcing secret key K: r_1, r_2, r_3, $\{t_1, t'_1, t_2, t'_2, z_1, z_2\} \in \mathbb{Z}_p$, $a_{j,i} \in \mathbb{Z}_p$, $(b_1, g_1^{b_1})$, $(b_2, g_1^{b_2})$, $(b_3, g_1^{b_3})$, $(b_4, g_1^{b_4})$, $(b_5, g_1^{b_5})$ and $(b_6, g_1^{b_6})$. Our outsourcing algorithm can be split into two steps. Firstly, client computes β_j and $\psi(\prod_{i \neq s} w_i^{a_{j,i}})$ with the help of cloud. After receiving answers from cloud, client computes their quotient then computes $\sigma_{j,s}$ with the help of cloud. We observe that public parameters are located in the base part, which means that we do not need to blind base part during the communication between client and server. We only blind private parameters which are used in the signature generation process. Therefore, our algorithm is more efficient than traditional algorithms. The communication between the client and the computation server can be carried out on an open channel.

The global parameters are (e, ψ, p, q, $\mathbb{G}_1$, $\mathbb{G}_2$, $\mathbb{G}_T$, g_1, g_2, H_1, H_2, h, d, n, k), which are mentioned in section 3.1. The inputs of **RS** include 4 parts:

1. All the d group members' public keys $(\mathrm{pk}_1, \mathrm{pk}_2, ..., \mathrm{pk}_d) = (w_1, w_2, ..., w_d)$.
2. A message block $m_j = (m_{j,1}, m_{j,2}, ..., m_{j,k})$ and its identifier id_j.
3. The public aggregate keys $\mathrm{pak} = (\eta_1, \eta_2, ..., \eta_k)$.
4. Client C's number s and his/her secret key sk_s.

The output of **RS** is signature $\sigma_j = (\sigma_{j,1}, ..., \sigma_{j,d})$ of block m_j.

In Algorithm 2, client computes β_j with the support of cloud.

After invoking Algorithm 2, client C obtains β_j. Similarly, client computes $\psi(\prod_{i \neq s} w_i^{a_{j,i}})$ by Algorithm 3.

Client C obtains β_j and $\psi(\prod_{i \neq s} w_i^{a_{j,i}})$ by invoking Algorithms 2 and 3. After that, client computes σ_j by Algorithm 4.

Note that C needs to choose 3 integers r_1, r_2 and r_3 to do some modular exponentiation computing. For the purpose of reducing the computational burden of client, they should be as small as possible. For example, r_1, r_2 and r_3 can be chosen in [2, 21], so that the client could do verification easily. t_1, t'_1, t_2, t'_2, z_1 and z_2 are elements in $\mathbb{Z}_p$. For the privacy requirement of inputs, they should be at least 64-bit long.

We present the completed algorithm that contains four parts (KeyGen, β_j-computing, $\psi(\prod_{i\neq s}w_i^{a_{j,i}})$-computing, σ_j-computing) as follows:

- KeyGen(1^κ). On input a security parameter κ, the client invokes Algorithm 1 to get an outsourcing secret key K: r_1, r_2, r_3, $\{t_1, t_1', t_2, t_2', z_1, z_2\}\in\mathbb{Z}_p$, $a_{j,i}\in\mathbb{Z}_p$, $(b_1, g_1^{b_1})$, $(b_2, g_1^{b_2})$, $(b_3, g_1^{b_3})$, $(b_4, g_1^{b_4})$, $(b_5, g_1^{b_5})$ and $(b_6, g_1^{b_6})$.
- β_j-computing(K, pak, m_j, id_j): On input outsourcing secret key, public aggregate keys, message block and its identifier, client outsources computation task of β_j to server S by invoking Algorithm 2.
- $\psi(\prod_{i\neq s}w_i^{a_{j,i}})$-computing($K$, w_1, w_2, ..., w_d, $a_{j,i}$): On input outsourcing secret key, public keys and random numbers chosen by user, client outsources computation task of $\psi(\prod_{i\neq s}w_i^{a_{j,i}})$ to server S by invoking Algorithm 3.
- σ_j-computing(K, sk_s, β_j, $\psi(\prod_{i\neq s}w_i^{a_{j,i}})$ and $a_{j,i}$): On input outsourcing secret key, secret key, result of Algorithms 2 and 3 and random numbers chosen by user, client outsources computation task of σ_j to server S by invoking Algorithm 4.

Algorithm 3: Algorithm of computing $\psi(\prod_{i\neq s}w_i^{a_{j,i}})$

Input: K, $(w_1, w_2, ..., w_d)$ and $a_{j,i}\in\mathbb{Z}_p$.

Output: "error" or $\psi(\prod_{i\neq s}w_i^{a_{j,i}})$.

1: Let $mul_j' = \prod_{i\neq s}w_i^{a_{j,i}}$, C computes v_i, v_i', θ_2 and θ_2' which satisfy $a_{j,i} = v_i + t_2\theta_2$ and $r_2a_{j,i} = v_i' + t_2'\theta_2'$, where $i = 1, 2, ..., d$ and $i \neq s$. After that, mul_j' and $mul_j''^{r_2}$ are divided as:

$$mul_j' = \prod\nolimits_{i\neq s} w_i^{a_{j,i}} = (\prod\nolimits_{i\neq s} w_i^{v_i})\cdot((\prod\nolimits_{i\neq s} w_i)^{\theta_2})^{t_2};$$

$$mul_j''^{r_2} = \prod\nolimits_{i\neq s} w_i^{r_2a_{j,i}} = (\prod\nolimits_{i\neq s} w_i^{v_i'})\cdot((\prod\nolimits_{i\neq s} w_i)^{\theta_2'})^{t_2'}.$$

2: Client C queries server S in random order as:

$$S(w_i, v_i)\rightarrow w_i^{v_i};$$
$$S(w_i, v_i')\rightarrow w_i^{v_i'};$$
$$S((\prod\nolimits_{i\neq s} w_i), \theta_2)\rightarrow(\prod\nolimits_{i\neq s} w_i)^{\theta_2};$$
$$S((\prod\nolimits_{i\neq s} w_i), \theta_2')\rightarrow(\prod\nolimits_{i\neq s} w_i)^{\theta_2'},$$

where $i = 1, 2, ..., d$ and $i \neq s$.

3: Server S computes them and returns the answer to client C.

4: Client C receives the answer from server S and checks the outputs. Client C checks whether the equation holds:

$$((\prod\nolimits_{i\neq s} w_i^{v_i})\cdot((\prod\nolimits_{i\neq s} w_i)^{\theta_2})^{t_2})^{r_2} = (\prod\nolimits_{i\neq s} w_i^{v_i'})\cdot((\prod\nolimits_{i\neq s} w_i)^{\theta_2'})^{t_2'}.$$

If not, C outputs "error". Otherwise, C computes

$$\psi(\prod\nolimits_{i\neq s} w_i^{a_{j,i}}) = \psi((\prod\nolimits_{i\neq s} w_i^{v_i})\cdot((\prod\nolimits_{i\neq s} w_i)^{\theta_2})^{t_2}).$$

Algorithm 4: Algorithm of computing σ_j

Input: K, sk_s, β_j, $\psi(\prod_{i\neq s} w_i^{a_{j,i}})$ and $a_{j,i}\in\mathbb{Z}_p$.

Output: "error" or σ_j.

1: Client C computes $base = \dfrac{\beta_j}{\psi(\prod_{i\neq s} w_i^{a_{j,i}})}$ and $pow = \frac{1}{\mathrm{sk}_s}$.

2: Client C uses the 6 blinding pairs generated in Algorithm 1 to blind $base$ and pow. Let δ_1, $g_1^{b_1} \bmod p$, $\delta_2 = g_1^{b_2} \bmod p$, $\varepsilon_1 = g_1^{b_5} \bmod p$ and $\varepsilon_2 = g_1^{b_6} \bmod p$. After that, Client C computes γ_1, y_1, γ_2 and y_2, which satisfy

$$\begin{cases} b_5 \cdot pow - b_1 z_1 y_1 = b_3 \\ pow = \gamma_1 + z_1 y_1 \\ b_6 \cdot r_3 \cdot pow - b_2 z_2 y_2 = b_4 \\ r_3 \cdot pow = \gamma_2 + z_2 y_2 \end{cases}$$

3: Client C computes $c_1 = base/\varepsilon_1$ and $c_2 = base/\varepsilon_2$.

After 2 and 3, $base^{pow}$ is transformed as follows:

$$\begin{aligned} base^{pow} &= (\varepsilon_1 c_1)^{pow} \\ &= g_1^{b_5 \cdot pow} c_1^{z_1 y_1} c_1^{\gamma_1} \\ &= g_1^{b_5 \cdot pow - b_1 z_1 y_1} g_1^{b_1 z_1 y_1} c_1^{z_1 y_1} c_1^{\gamma_1}. \\ &= g_1^{b_3} \delta_1^{z_1 y_1} c_1^{z_1 y_1} c_1^{\gamma_1} \\ &= g_1^{b_3} c_1^{\gamma_1} ((\delta_1 c_1)^{z_1})^{y_1} \end{aligned}$$

Similarly, $base^{r_3 \cdot pow}$ is transformed as $base^{r_3 \cdot pow} = g_1^{b_4} c_2^{\gamma_2} ((\delta_2 c_2)^{z_2})^{y_2}$.

4: Client C queries server S in random order as:

$$\begin{aligned} &S(c_1, \gamma_1) \to c_1^{\gamma_1} \\ &S(c_2, \gamma_2) \to c_2^{\gamma_2} \\ &S((\delta_1 c_1), y_1) \to (\delta_1 c_1)^{y_1} \\ &S((\delta_2 c_2), y_2) \to (\delta_2 c_2)^{y_2} \end{aligned}$$

5: Server S computes them and returns the answer to Client C.

6: Client C receives the answer from server S and checks the outputs. Client C checks whether the equation holds:

$$(g_1^{b_3} c_1^{\gamma_1} ((\delta_1 c_1)^{y_1})^{z_1})^{r_3} = g_1^{b_4} c_2^{\gamma_2} ((\delta_2 c_2)^{y_2})^{z_2}.$$

If not, C outputs "error". Otherwise, C computes

$$\sigma_{j,s} = base^{pow} = g_1^{b_3} c_1^{\gamma_1} ((\delta_1 c_1)^{z_1})^{y_1}.$$

7: Client C computes $\sigma_{j,i} = g_1^{a_{j,i}}$ for $i \neq s$. Then C gets $\sigma_j = (\sigma_{j,1}, \ldots, \sigma_{j,d})$.

4. SECURITY ANALYSIS

4.1 Privacy Analysis

Theorem 1: In the fully malicious model, the algorithm **RS** is private for inputs including $m_j=(m_{j,1}, m_{j,2}, ..., m_{j,k})$, secret key sk_s and $a_{j,i}\in\mathbb{Z}_p$ randomly chosen by C.

***Proof*:** In the algorithm **RS**, the message block $m_j = (m_{j,1}, m_{j,2}, ..., m_{j,k})$, secret key sk_s and $a_{j,i}\in\mathbb{Z}_p$ should keep secret.

We first prove the privacy for input $m_j=(m_{j,1}, m_{j,2}, ..., m_{j,k})$. Note that the message block $m_j = (m_{j,1}, m_{j,2}, ..., m_{j,k})$ is transformed as $m_{j,l}=u_l + t_1\theta_1$ in Algorithm 2, and the adversary $\mathcal{A}$ can only know u_l and θ_1 throughout the whole algorithm **RS**. $\mathcal{A}$ is asked to guess a 64-bit long integer to compute $m'_{j,l}$, and the probability that $\mathcal{A}$ guesses t_1 correctly and recovers $m_{j,l}$ is negligible. $m_{j,l}$ is blinded by t_1 in the sense of indistinguishability.

We then prove the privacy for $a_{j,i}\in\mathbb{Z}_p$. Similarly, $a_{j,i}$ is transformed as $a_{j,i} = v_i+t_2\theta_2$ in Algorithm 3, and the adversary $\mathcal{A}$ can only know v_i throughout the whole algorithm **RS**. $\mathcal{A}$ could only figure out $a_{j,i}$ throughout guessing t_2 correctly. t_2 is 64-bit long integer randomly chosen by C. Probability that $\mathcal{A}$ recovers $a_{j,i}$ is negligible. $a_{j,i}$ is blinded by t_2 in the sense of indistinguishability.

In Algorithm 4, *pow* is the reciprocal of secret key sk_s. *pow* is sum of 2 parts. One part is γ_1, and another part is z_1y_1. z_1 are kept secret by C. $\mathcal{A}$ can only know γ_1 and y_1 throughout **RS**. Since z_1 is a random blinding integer in $\mathbb{Z}_p$, *pow* is blinded by z_1 in the sense of indistinguishability. Besides, equations $base^{r_3\cdot pow} = g_1^{b_4}c_2^{\gamma_2}((\delta_2 c_2)^{z_2})^{y_2}$ and $base^{pow} = g_1^{b_3}c_1^{\gamma_1}((\delta_1 c_1)^{z_1})^{y_1}$ show that *base* is blinded by $g_1^{b_3}$ and $g_1^{b_4}$ which are kept secret by C and therefore $\mathcal{A}$ cannot work out *base*. Further, $\mathcal{A}$ cannot recover $m_j=(m_{j,1}, m_{j,2}, ..., m_{j,k})$ and $a_{j,i}$.

From the above, **RS** is private for inputs including $m_j=(m_{j,1}, m_{j,2}, ..., m_{j,k})$, secret key sk_s and $a_{j,i}$.

Theorem 2: In the fully malicious model, the algorithm **RS** is private for $\prod_{l=1}^{k}\eta_l^{m_{j,l}}$, $\prod_{i\neq s}w_i^{a_{j,i}}$ and $\sigma_{j,s}$.

***Proof*:** We first prove the privacy for $\prod_{l=1}^{k}\eta_l^{m_{j,l}}$. In Algorithm 2, C computes

$$\prod_{l=1}^{k}\eta_l^{m_{j,l}} = (\prod_{l=1}^{k}\eta_l^{u_l})\cdot((\eta_1\eta_2...\eta_k)^{\theta_1})^{t_1},$$

and the adversary $\mathcal{A}$ can only know $\eta_l^{u_l}$ and $(\eta_1\ \eta_2\ \dots\ \eta_k)^{\theta_1}$ throughout **RS**. $\mathcal{A}$ could not figure out $\prod_{l=1}^{k}\eta_l^{m_{j,l}}$ because t_1 is unknown to $\mathcal{A}$. If $\mathcal{A}$ correctly guesses t_1 (named $t_{1\mathcal{A}}$) and figure out $output_{\mathcal{A}} = (\prod_{l=1}^{k}\eta_l^{u_l})\cdot((\eta_1\eta_2...\eta_k)^{\theta_1})^{t_{1\mathcal{A}}}$, then $\prod_{l=1}^{k}\eta_l^{m_{j,l}}$ and $output_{\mathcal{A}}$ are indistinguishable for $\mathcal{A}$.

In Algorithm 3, C computes

$$\prod_{i\neq s}w_i^{a_{j,i}} = (\prod_{i\neq s}w_i^{v_i})\cdot((\prod_{i\neq s}w_i)^{\theta_2})^{t_2},$$

and the adversary $\mathcal{A}$ can only know $w_i^{v_i}$ and $(\prod_{i\neq s}w_i^{a_{j,i}})^{\theta_2}$ throughout **RS**. It is similar to the

situation in Algorithm 2 that $\mathscr{A}$could not figure out $\prod_{i\neq s} w_i^{a_{j,i}}$ because t_2 is unknown to $\mathscr{A}$. If $\mathscr{A}$guesses t_2 (named $t_{2\mathscr{A}}$) and figure out $output_{\mathscr{A}} = (\prod_{i\neq s} w_i^{y_i}) \cdot (\prod_{i\neq s} w_i)^{\theta_2})^{t_{2\mathscr{A}}}$ then $\prod_{i\neq s} w_i^{a_{j,i}}$ and $output_{\mathscr{A}}$ are indistinguishable for $\mathscr{A}$.

In Algorithm 4, C computes

$$\sigma_{j,s} = base^{pow} = g_1^{b_3} c_1^{\gamma_1} ((\delta_1 c_1)^{z_1})^{y_1},$$

and the adversary $\mathscr{A}$can only know $c_1^{\gamma_1}$ and $(\delta_1 c_1)^{z_1}$ throughout **RS**. $\mathscr{A}$could not figure out $\sigma_{j,s}$ because $g_1^{b_3}$ and y_1 are unknown to $\mathscr{A}$. If $\mathscr{A}$guesses $g_1^{b_3}$ and y_1 (named $g_{1\mathscr{A}}^{b_3}$ and $y_{1\mathscr{A}}$) correctly and figure out $output_{\mathscr{A}} = g_{1\mathscr{A}}^{b_3} c_1^{\gamma_1} ((\delta_1 c_1)^{z_1})^{y_{1\mathscr{A}}}$, then $\sigma_{j,s}$ and $output_{\mathscr{A}}$ are indistinguishable for $\mathscr{A}$.

Above all, the algorithm **RS** is privacy for $\prod_{l=1}^{k} \eta_l^{m_{j,l}}$, $\prod_{i\neq s} w_i^{a_{j,i}}$ and $\sigma_{j,s}$.

4.2 Checkability Analysis

Theorem 3: RS is $(1-\frac{k^2}{20(4k^2+6k+2)})(1-\frac{(d-1)^2}{40d(2d-1)})(1-\frac{1}{240})$-checkable according to Definition 4.

***Proof*:** Considering the page limitation, we omit the detailed proof here.

5. PERFORMANCE EVALUATION

5.1 Theoretical evaluation

In **RS**, the running time on the cloud side is very little, even negligible, because the computing resources of cloud server are almost unlimited. Hence, we focus on evaluating the client-side cost in our algorithm and other existing algorithms. For convenience, we use "Alg. 0" to denote the algorithm that client generates the signature by itself. In Alg. 0, all computation tasks are finished on the client-side. In **RS** and other existing secure outsourcing algorithms, most of client-side computing resources are spent on performing modular exponentiations and modular multiplications. Hence, we regard the numbers of modular exponentiations and modular multiplications as important indicators for theoretical evaluation about efficiency. Table 1 compares **RS** with other existing secure outsourcing algorithms in term of the numbers of modular exponentiations and modular multiplications need to be performed on the client-side. Let ME and MM denote modular exponentiation and modular multiplication respectively. We omit other operations such as modular additions because they cost very little time.

Table 1. Comparison of RS, MExp in [17], the algorithm in [16] and Alg. 0.

β_j computing				
	RS	**MExp** in [17]	algorithm in [16]	Alg. 0
ME	3	3	1	k
MM	$3k+2$	$7k+10$	$6k+6$	k
ψ computing				
	RS	**MExp** in [17]	algorithm in [16]	Alg. 0
ME	3	3	1	$d-1$
MM	$3d-2$	$7d+2$	$6d-1$	$d-2$

If we use **MExp** in [17] to do modular exponentiations, we have to spend extra computing resources to protect the base privacy of modular exponentiations in the signature generation process. Actually, they are public parameters and we do not need to protect the privacy of them. Table 1 shows that the number of modular multiplications in MExp is two times more than that in **RS**. Therefore, compared with **MExp**, **RS** is more efficient. If we use the algorithm proposed in [16] to do modular exponentiations, we also protect the privacy of the bases. Compare with our algorithm, the algorithm proposed in [16] also contains constant times of modular exponentiations, but it contains much more modular multiplications. Besides, the checkability of the algorithm in [16] is not satisfying. Specifically, if the server cheats the client in the process of computing β_j, the probability that the client can detect the misbehavior is only $\frac{1}{k+1}$. This means that the malicious cloud server has a high probability of cheating clients. Therefore, our algorithm is more suitable than other existing algorithms for this scenario.

5.2 Experimental Evaluation

In this section, we provide an experimental evaluation of **RS** to show the performance of our scheme. Similar to Section 5.1, we use "Alg. 0" to denote the algorithm that client generates the signature by itself. In Alg. 0, all computation tasks are finished on the client-side. We show the efficiency of our algorithm by comparing the time consumption of **RS** and Alg. 0. These experiments are carried out on a computer with an Intel Pentium processor of 2.70 GHz and 4 GB memory. The operation system version is Ubuntu 16.04. We use C language based on the free Pairing-Based Cryptography (PBC) Library and the GNU Multiple Precision Arithmetic (GMP). In these experiments, we set the base field size to be 512 bits and the size of an element in $\mathbb{Z}_p^*$ to be 160 bits.

Firstly, we compare the computation time in computing β_j between Alg. 0 and **RS**. In Fig. 2, the number of blocks and the number of sections vary from 10 to 100. From Fig. 2, we could observe that the computation time in **RS** is obviously less than the time in Alg. 0. When the number of blocks and the number of sections are both 80, **RS** costs about 0.43s while Alg. 0 costs about 11s.

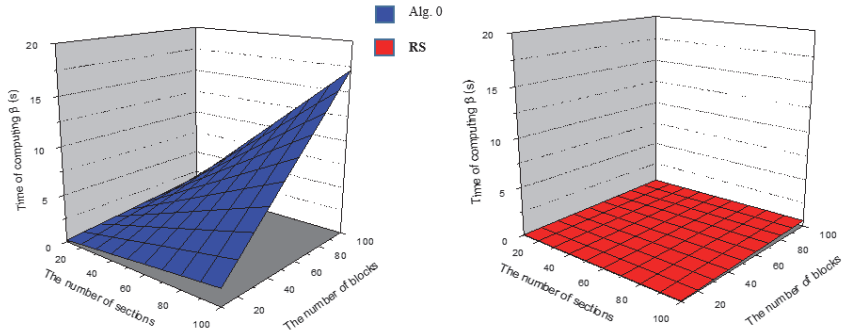


Fig. 2. Computation time of β.

Secondly, Fig. 3 shows the computation time in computing $\psi(\prod_{i\neq s} w_i^{a_{j,i}})$ between Alg. 0 and **RS**. Similarly, the number of blocks and the number of group members vary from

10 to 100. From Fig. 3, we could learn that **RS** still has better performance than Alg. 0. When the number of blocks and the number of group members are both 100, **RS** costs about 0.58s while Alg. 0 costs about 17.04s.

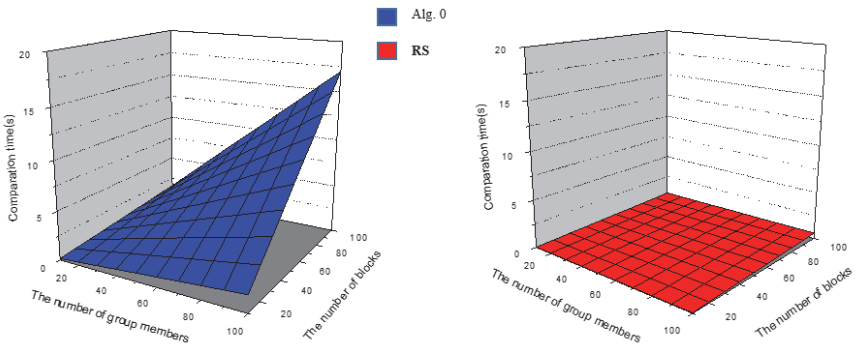


Fig. 3. Computation time of ψ.

Thirdly, we can see the time of computing σ_i between Alg. 0 and **RS** in Fig. 4. The number of blocks varies from 10 to 100. **RS** has ordinary performance at this stage. When the number of blocks is 100, **RS** costs about 0.808s to compute while Alg. 0 costs about 0.959s.

Finally, the comparing of total computing time is shown in Fig. 5. The number of group members and the number of sections of each block are both 100, and the number of blocks varies from 10 to 100. When the number of blocks is 100, **RS** costs about 1.98s while Alg. 0 costs about 35.18s. The client-side cost in **RS** is reduced to 5.6% of that in Alg. 0. In general, experiment results proved the efficiency of our algorithm.

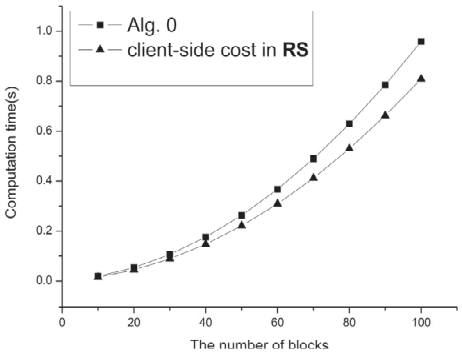


Fig. 4. Computation time of σ.

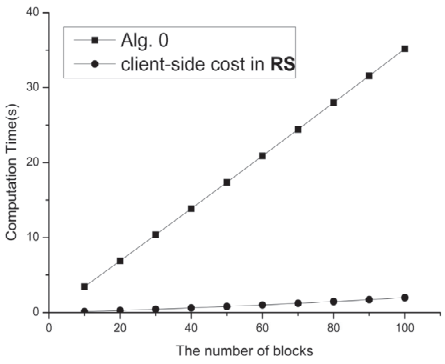


Fig. 5. Total computation time.

6. CONCLUSION

Ring signature generation is the most complex part in public auditing. Sometimes it is the bottleneck of efficiency improving in public auditing schemes. In this paper, we

introduce a secure outsourcing algorithm for ring signature generation with high performance in efficiency and checkability. The privacy of input and output could be guaranteed in our algorithm, and we give formal proofs of security. Experimental evaluation also shows the good performance in efficiency improving. In conclusion, our algorithm makes public auditing more practical for the users with resource-constraint devices.

ACKNOWLEDGEMENT

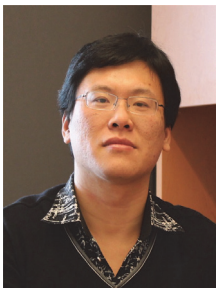
This research is supported by National Natural Science Foundation of China (61572267), National Development Foundation of Cryptography (MMJJ20170118), the Open Project of the State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Sciences.

REFERENCES

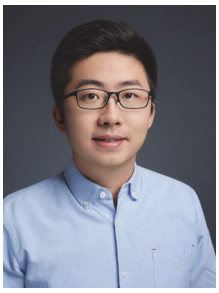
1. K. Ren, C. Wang, and Q. Wang, "Security challenges for the public cloud," *IEEE Internet Computing*, Vol. 16, 2012, pp. 69-73.
2. C. Wang, Q. Wang, K. Ren, and W. Lou, "Privacy-preserving public auditing for data storage security in cloud computing," in *Proceedings of IEEE Conference on Computer Communications*, Vol. 62, 2010, pp. 525-533.
3. B. Wang, B. Li, and H. Li, "Oruta: Privacy-preserving public auditing for shared data in the cloud," in *Proceedings of IEEE 5th International Conference on Cloud Computing*, 2012, pp. 295-302.
4. G. Ateniese, R. Burns, R. Curtmola, J. Herring, L. Kissner, Z. Peterson, and D. Song, "Provable data possession at untrusted stores," in *Proceedings of ACM Conference on Computer and Communications Security*, 2007, pp. 598-609.
5. R. L. Rivest, A. Shamir, and Y. Tauman, "How to leak a secret," in *Proceedings of the 7th International Conference on the Theory and Application of Cryptology and Information Security: Advances in Cryptology*, 2001, pp. 554-567.
6. A. Juels and B. S. Kaliski, "Pors: proofs of retrievability for large files," in *Proceedings of ACM Conference on Computer and Communications Security*, 2007, pp. 584-597.
7. H. Shacham and B. Waters, "Compact proofs of retrievability," *Journal of Cryptology*, Vol. 26, 2008, pp. 442-483.
8. J. Yu, K. Ren, C. Wang, and V. Varadharajan, "Enabling cloud storage auditing with key-exposure resistance," *IEEE Transactions on Information Forensics and Security*, Vol. 10, 2017, pp. 1167-1179.
9. J. Yu and H. Q. Wang, "Strong key-exposure resilient auditing for secure cloud storage," *IEEE Transactions on Information Forensics and Security*, Vol. 12, 2017, pp. 1931-1940.
10. J. Yu, R. Hao, H. Xia, H. Zhang, X. Cheng, and F. Kong, "Intrusion-resilient identitybased signatures: Concrete scheme in the standard model and generic construction," *Information Sciences*, Vol. 442, 2018, pp. 158-172.
11. Y. Zhang, J. Yu, R. Hao, C. Wang, and K. Ren, "Enabling efficient user revocation

in identity-based cloud storage auditing for shared big data," *IEEE Transactions on Dependable and Secure Computing*, 2018, to appear.
12. W. Shen, J. Qin, J. Yu, R. Hao, and J. Hu, "Enabling identity-based integrity auditing and data sharing with sensitive information hiding for secure cloud storage," *IEEE Transactions on Information Forensics and Security*, Vol. 14, 2019, pp. 331-346.
13. D. Chaum and T. P. Pedersen, "Wallet databases with observers," in *Proceedings of International Cryptology Conference*, 1992, pp. 89-105.
14. S. Hohenberger and A. Lysyanskaya, "How to securely outsource cryptographic computations," *Theory of Cryptography*, Vol. 3378, 2005, pp. 264-282.
15. X. Chen, J. Li, J. Ma, Q. Tang, and W. Lou, "New algorithms for secure outsourcing of modular exponentiations," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 25, 2014, pp. 2386-2396.
16. Y. Wang, Q. Wu, D. S. Wong, B. Qin, S. S. M. Chow, Z. Liu, and X. Tan, "Securely outsourcing exponentiations with single untrusted program for cloud storage," in *Proceedings of European Symposium on Research in Computer Security*, 2014, pp. 326-343.
17. Y. Ding, Z. Xu, J. Ye, and K. K. R. Choo, "Secure outsourcing of modular exponentiations under single untrusted programme model," *Journal of Computer and System Sciences*, Vol. 90, 2017, pp. 1-13.
18. X. Chen, X. Huang, J. Li, J. Ma, W. Lou, and D. S. Wong, "New algorithms for secure outsourcing of large-scale systems of linear equations," *IEEE Transactions on Information Forensics and Security*, Vol. 10, 2014, pp. 69-78.
19. H. Tian, F. Zhang, and K. Ren, "Secure bilinear pairing outsourcing made more efficient and flexible," in *Proceedings of ACM Symposium on Information*, *Computer and Communications Security*, 2015, pp. 417-426.
20. X. Lei, X. Liao, T. Huang, and H. Li, "Cloud computing service: The case of large matrix determinant computation," *IEEE Transactions on Services Computing*, Vol. 8, 2015, pp. 688-700.
21. J. Yu, K. Ren, and C. Wang, "Enabling cloud storage auditing with verifiable outsourcing of key updates," *IEEE Transactions on Information Forensics and Security*, Vol. 11, 2016, pp. 1362-1375.
22. R. Gennaro, C. Gentry, and B. Parno, "Non-interactive verifiable computing: Outsourcing computation to untrusted workers," in *Proceedings of Conference on Advances in Cryptology*, 2010, pp. 465-482.

Pu Zhao received the B.Eng. degree in Information Security from Qingdao University in 2016, where he is currently pursuing the master degree in Computer Science and Technology. His research interests include cloud computing security, secure outsourcing computation, and graph encryption.



Jia Yu received the B.S. and M.S. degrees from the School of Computer Science and Technology, and the Ph.D. degree from the Institute of Network Security Shandong University, in 2003, 2000, and 2006, respectively. He is currently a Professor with the College of Computer Science and Technology, Qingdao University. His research interests include cloud computing security, key evolving cryptography, digital signature, and network security.



Hanlin Zhang received the B.S. degree in Software Engineering from Qingdao University in 2010, and the M.S. degree in Applied Information Technology and the Ph.D. degree in Information Technology from Towson University, MD, USA, in 2011 and 2016, respectively. He is currently with the College of Computer Science and Technology, Qingdao University, as an Assistant Professor. His research interests include information security, cloud security, mobile security, and network security.