# Oruta: Privacy-Preserving Public Auditing for Shared Data in the Cloud

Boyang Wang, *Student Member, IEEE*, Baochun Li, *Senior Member, IEEE*, and Hui Li, *Member, IEEE*

**Abstract**—With cloud data services, it is commonplace for data to be not only stored in the cloud, but also shared across multiple users. Unfortunately, the integrity of cloud data is subject to skepticism due to the existence of hardware/software failures and human errors. Several mechanisms have been designed to allow both data owners and public verifiers to efficiently audit cloud data integrity without retrieving the entire data from the cloud server. However, public auditing on the integrity of shared data with these existing mechanisms will inevitably reveal confidential information—identity privacy—to public verifiers. In this paper, we propose a novel privacy-preserving mechanism that supports public auditing on shared data stored in the cloud. In particular, we exploit ring signatures to compute verification metadata needed to audit the correctness of shared data. With our mechanism, the identity of the signer on each block in shared data is kept private from public verifiers, who are able to efficiently verify shared data integrity without retrieving the entire file. In addition, our mechanism is able to perform multiple auditing tasks simultaneously instead of verifying them one by one. Our experimental results demonstrate the effectiveness and efficiency of our mechanism when auditing shared data integrity.

**Index Terms**—Public auditing, privacy-preserving, shared data, cloud computing

## 1 Introduction

Cloud service providers offer users efficient and scalable data storage services with a much lower marginal cost than traditional approaches [2]. It is routine for users to leverage cloud storage services to share data with others in a group, as data sharing becomes a standard feature in most cloud storage offerings, including Dropbox, iCloud and Google Drive.

The integrity of data in cloud storage, however, is subject to skepticism and scrutiny, as data stored in the cloud can easily be lost or corrupted due to the inevitable hardware/software failures and human errors [3], [4]. To make this matter even worse, cloud service providers may be reluctant to inform users about these data errors in order to maintain the reputation of their services and avoid losing profits [5]. Therefore, the integrity of cloud data should be verified before any data utilization, such as search or computation over cloud data [6].

The traditional approach for checking data correctness is to retrieve the entire data from the cloud, and then verify data integrity by checking the correctness of signatures (e.g., RSA [7]) or hash values (e.g., MD5 [8]) of the entire data. Certainly, this conventional approach is able to successfully check the correctness of cloud data. However, the efficiency of using this traditional approach on cloud data is in doubt [9].

The main reason is that the size of cloud data is large in general. Downloading the entire cloud data to verify data integrity will cost or even waste users amounts of computation and communication resources, especially when data have been corrupted in the cloud. Besides, many uses of cloud data (e.g., data mining and machine learning) do not necessarily need users to download the entire cloud data to local devices [2]. It is because cloud providers, such as Amazon, can offer users computation services directly on large-scale data that already existed in the cloud.

Recently, many mechanisms [9], [10], [11], [12], [13], [14], [15], [16], [17] have been proposed to allow not only a data owner itself but also a *public verifier* to efficiently perform integrity checking without downloading the entire data from the cloud, which is referred to as *public auditing* [5]. In these mechanisms, data is divided into many small blocks, where each block is independently signed by the owner; and a random combination of all the blocks instead of the whole data is retrieved during integrity checking [9]. A public verifier could be a data user (e.g., researcher) who would like to utilize the owner's data via the cloud or a third-party auditor (TPA) who can provide expert integrity checking services [18]. Moving a step forward, Wang et al. designed an advanced auditing mechanism [5] (named as WWRL in this paper), so that during public auditing on cloud data, the content of private data belonging to a personal user is not disclosed to any public verifiers. Unfortunately, current public auditing solutions mentioned above only focus on personal data in the cloud [1].

We believe that sharing data among multiple users is perhaps one of the most engaging features that motivates cloud storage. Therefore, it is also necessary to ensure the integrity of shared data in the cloud is correct. Existing

- *B. Wang is with the State Key Laboratory of Integrated Service Networks, Xidian University, Xi'an, 710071, China. E-mail: bywang@mail.xidian.edu.cn.*
- *B. Li is with the Department of Electrical and Computer Engineering, University of Toronto, Toronto, ON M5S 3G4, Canada. E-mail: bli@eecg.toronto.edu.*
- *H. Li is with the State Key Laboratory of Integrated Service Networks, Xidian University, Xi'an, 710071, China. E-mail: lihui@mail.xidian.edu.cn.*

*Manuscript received 3 Mar. 2013; revised 13 Dec. 2013; accepted 2 Jan. 2014; date of publication 12 Jan. 2014; date of current version 30 Apr. 2014.*
*Recommended for acceptance by A. Martin.*
*For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.*
*Digital Object Identifier no. 10.1109/TCC.2014.2299807*

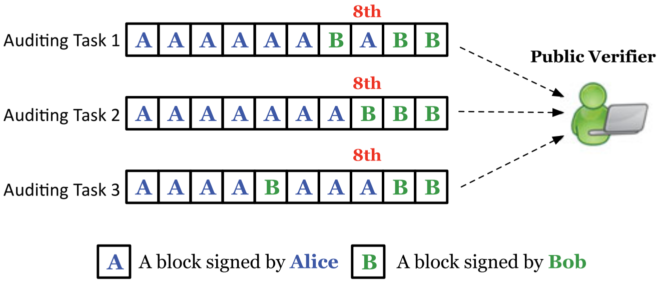

8th
Auditing Task 1 A A A A A A B A B B
8th
Auditing Task 2 A A A A A A A B B B
8th
Auditing Task 3 A A A A B A A A B B
Public Verifier
A A block signed by Alice
B A block signed by Bob

Fig. 1. Alice and Bob share a data file in the cloud, and a public verifier audits shared data integrity with existing mechanisms.

public auditing mechanisms can actually be extended to verify shared data integrity [1], [5], [19], [20]. However, a new significant privacy issue introduced in the case of shared data with the use of existing mechanisms is the leakage of *identity privacy* to public verifiers [1].

For instance, Alice and Bob work together as a group and share a file in the cloud (as presented in Fig. 1). The shared file is divided into a number of small blocks, where each block is independently signed by one of the two users with existing public auditing solutions (e.g., [5]). Once a block in this shared file is modified by a user, this user needs to sign the new block using his/her private key. Eventually, different blocks are signed by different users due to the modification introduced by these two different users. Then, in order to correctly audit the integrity of the entire data, a public verifier needs to choose the appropriate public key for each block (e.g., a block signed by Alice can only be correctly verified by Alice's public key). As a result, this public verifier will inevitably learn the identity of the signer on each block due to the unique binding between an identity and a public key via digital certificates under public key infrastructure (PKI).

Failing to preserve identity privacy on shared data during public auditing will reveal significant confidential information (e.g., which particular user in the group or special block in shared data is a more valuable target) to public verifiers. Specifically, as shown in Fig. 1, after performing several auditing tasks, this public verifier can first learn that Alice may be a more important role in the group because most of the blocks in the shared file are always signed by Alice; on the other hand, this public verifier can also easily deduce that the eighth block may contain data of a higher value (e.g., a final bid in an auction), because this block is frequently modified by the two different users. In order to protect these confidential information, it is essential and critical to preserve identity privacy from public verifiers during public auditing.

In this paper, to solve the above privacy issue on shared data, we propose Oruta,[1] a novel privacy-preserving public auditing mechanism. More specifically, we utilize ring signatures [21] to construct homomorphic authenticators [10] in Oruta, so that a public verifier is able to verify the integrity of shared data without retrieving the entire data—while the identity of the signer on each block in shared data is kept private from the public verifier. In addition, we further extend our mechanism to support batch auditing, which can perform multiple auditing tasks simultaneously and improve the efficiency of verification for multiple auditing tasks. Meanwhile, Oruta is compatible with random masking [5], which has been utilized in WWRL and can preserve data privacy from public verifiers. Moreover, we also leverage index hash tables from a previous public auditing solution [15] to support dynamic data. A high-level comparison among Oruta and existing mechanisms is presented in Table 1.

TABLE 1
Comparison among Different Mechanisms

| | PDP [9] | WWRL [5] | Oruta |
|---|---|---|---|
| Public Auditing | √ | √ | √ |
| Data Privacy | × | √ | √ |
| Identity Privacy | × | × | √ |

The remainder of this paper is organized as follows. In Section 2, we present the system model, threat model and design objectives. In Section 3, we introduce cryptographic primitives used in Oruta. The detailed design and security analysis of Oruta are presented in Section 4 and Section 5. In Section 6, we evaluate the performance of Oruta. Finally, we briefly discuss related work in Section 7, and conclude this paper in Section 8.

# 2 Problem Statement

## 2.1 System Model

As illustrated in Fig. 2, the system model in this paper involves three parties: the cloud server, a group of users and a public verifier. There are two types of users in a group: the original user and a number of group users. The original user initially creates shared data in the cloud, and shares it with group users. Both the original user and group users are members of the group. Every member of the group is allowed to access and modify shared data. Shared data and its verification metadata (i.e., signatures) are both stored in the cloud server. A public verifier, such as a third-party auditor providing expert data auditing services or a data user outside the group intending to utilize shared data, is able to publicly verify the integrity of shared data stored in the cloud server.

When a public verifier wishes to check the integrity of shared data, it first sends an auditing challenge to the cloud server. After receiving the auditing challenge, the

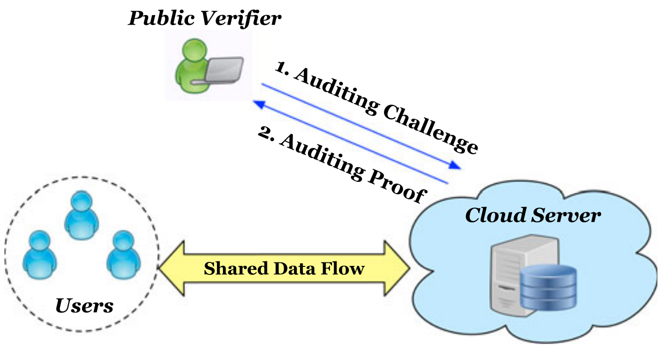


Fig. 2. Our system model includes the cloud server, a group of users and a public verifier.

[1] Oruta stands for "One Ring to Rule Them All."

cloud server responds to the public verifier with an auditing proof of the possession of shared data. Then, this public verifier checks the correctness of the entire data by verifying the correctness of the auditing proof. Essentially, the process of public auditing is a challenge-and-response protocol between a public verifier and the cloud server [9].

### 2.2 Threat Model

*Integrity Threats.* Two kinds of threats related to the integrity of shared data are possible. First, an adversary may try to corrupt the integrity of shared data. Second, the cloud service provider may inadvertently corrupt (or even remove) data in its storage due to hardware failures and human errors. Making matters worse, the cloud service provider is economically motivated, which means it may be reluctant to inform users about such corruption of data in order to save its reputation and avoid losing profits of its services.

*Privacy Threats.* The identity of the signer on each block in shared data is private and confidential to the group. During the process of auditing, a public verifier, who is only allowed to verify the correctness of shared data integrity, may try to reveal the identity of the signer on each block in shared data based on verification metadata. Once the public verifier reveals the identity of the signer on each block, it can easily distinguish a high-value target (a particular user in the group or a special block in shared data) from others.

### 2.3 Design Objectives

Our mechanism, Oruta, should be designed to achieve following properties: (1) *Public Auditing*: A public verifier is able to publicly verify the integrity of shared data without retrieving the entire data from the cloud. (2) *Correctness*: A public verifier is able to correctly verify shared data integrity. (3) *Unforgeability*: Only a user in the group can generate valid verification metadata (i.e., signatures) on shared data. (4) *Identity Privacy*: A public verifier cannot distinguish the identity of the signer on each block in shared data during the process of auditing.

### 2.4 Possible Alternative Approaches

To preserve the identity of the signer on each block during public auditing, one possible alternative approach is to ask all the users of the group to share a *global private key* [22], [23]. Then, every user is able to sign blocks with this global private key. However, once one user of the group is compromised or leaving the group, a new global private key must be generated and securely shared among the rest of the group, which clearly introduces huge overhead to users in terms of key management and key distribution. While in our solution, each user in the rest of the group can still utilize its own private key for computing verification metadata without generating or sharing any new secret keys.

Another possible approach to achieve identity privacy, is to add a trusted proxy between a group of users and the cloud in the system model. More concretely, each member's data is collected, signed, and uploaded to the cloud by this trusted proxy, then a public verifier can only verify and learn that it is the proxy signs the data, but cannot learn the identities of group members. Yet, the security of this method is threatened by the single point failure of the proxy. Besides, sometimes, not all the group members would like to trust the same proxy for generating signatures and uploading data on their behalf. Utilizing group signatures [24] is also an alternative option to preserve identity privacy. Unfortunately, as shown in our recent work [25], how to design an efficient public auditing mechanism based on group signatures remains open.[2]

Trusted Computing offers another possible alternative approach to achieve the design objectives of our mechanism. Specifically, by utilizing direct anonymous attestation [26], which is adopted by the Trusted Computing Group as the anonymous method for remote authentication in trusted platform module, users are able to preserve their identity privacy on shared data from a public verifier. The main problem with this approach is that it requires all the users using designed hardware, and needs the cloud provider to move all the existing cloud services to the trusted computing environment, which would be costly and impractical.

## 3 Preliminaries

In this section, we briefly introduce cryptographic primitives and their corresponding properties that we implement in Oruta.

### 3.1 Bilinear Maps

Let $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$ be three multiplicative cyclic groups of prime order $p$, $g_1$ be a generator of $\mathbb{G}_1$, and $g_2$ be a generator of $\mathbb{G}_2$. A bilinear map $e$ is a map $e$: $\mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ with the following properties:

- *Computability*: there exists an efficiently computable algorithm for computing map $e$.
- *Bilinearity*: for all $u \in \mathbb{G}_1$, $v \in \mathbb{G}_2$ and $a, b \in \mathbb{Z}_p$, $e(u^a, v^b) = e(u, v)^{ab}$.
- *Non-degeneracy*: $e(g_1, g_2) \neq 1$.

Bilinear maps can be generally constructed from certain elliptic curves [27]. Readers do not need to learn the technical details about how to build bilinear maps from certain elliptic curves. Understanding the properties of bilinear maps described above is sufficient enough for readers to access the design of our mechanism.

### 3.2 Security Assumptions

The security of our proposed mechanism is based on the two following assumptions:

*Computational Co-Diffie-Hellman (Co-CDH) Problem.* Let $a \in \mathbb{Z}_p^*$, given $g_2, g_2^a \in \mathbb{G}_2$ and $h \in \mathbb{G}_1$ as input, output $h^a \in \mathbb{G}_1$.

**Definition 1 (Computational Co-Diffie-Hellman Assumption).** *The advantage of a probabilistic polynomial time*

[2] The direct leverage of group signatures in an public auditing mechanism makes the size of verification metadata extremely huge, which is much larger than the size of data itself. See [25] for details.

*algorithm $\mathcal{A}$ in solving the Co-CDH problem on $(\mathbb{G}_1, \mathbb{G}_2)$ is defined as*

$$\mathtt{AdvCoCDH}_{\mathcal{A}} = \Pr\big[\mathcal{A}\big(g_2, g_2^a, h\big) = h^a : a \overset{R}{\leftarrow} \mathbb{Z}_p^*, h \overset{R}{\leftarrow} \mathbb{G}_1\big],$$

*where the probability is over the choice of $a$ and $h$, and the coin tosses of $\mathcal{A}$. The Co-CDH assumption means, for any probabilistic polynomial time algorithm $\mathcal{A}$, the advantage of it in solving the Co-CDH problem on $(\mathbb{G}_1, \mathbb{G}_2)$ is negligible.*

For the ease of understanding, we can also say solving the Co-CDH problem on $(\mathbb{G}_1, \mathbb{G}_2)$ is or computationally infeasible or hard under the Co-CDH assumption.

*Discrete Logarithm (DL) Problem.* Let $a \in \mathbb{Z}_p^*$, given $g, g^a \in \mathbb{G}_1$ as input, output $a$.

**Definition 2 (Discrete Logarithm Assumption).** *The advantage of a probabilistic polynomial time algorithm $\mathcal{A}$ in solving the DL problem in $\mathbb{G}_1$ is defined as*

$$\mathtt{AdvDL}_{\mathcal{A}} = \Pr\big[\mathcal{A}(g, g^a) = a : a \overset{R}{\leftarrow} \mathbb{Z}_p^*\big],$$

*where the probability is over the choice of $a$, and the coin tosses of $\mathcal{A}$. The DL Assumption means, for any probabilistic polynomial time algorithm $\mathcal{A}$, the advantage of it in solving the DL problem in $\mathbb{G}_1$ is negligible.*

### 3.3 Ring Signatures

The concept of ring signatures was first proposed by Rivest et al. [28] in 2001. With ring signatures, a verifier is convinced that a signature is computed using one of group members' private keys, but the verifier is not able to determine which one. More concretely, given a ring signature and a group of $d$ users, a verifier cannot distinguish the signer's identity with a probability more than $1/d$. This property can be used to preserve the identity of the signer from a verifier.

The ring signature scheme introduced by Boneh et al. [21] (referred to as BGLS in this paper) is constructed on bilinear maps. We will extend this ring signature scheme to construct our public auditing mechanism.

### 3.4 Homomorphic Authenticators

Homomorphic authenticators (also called homomorphic verifiable tags) are basic tools to construct public auditing mechanisms [1], [5], [9], [10], [12], [15]. Besides unforgeability (i.e., only a user with a private key can generate valid signatures), a homomorphic authenticable signature scheme, which denotes a homomorphic authenticator based on signatures, should also satisfy the following properties:

Let $(\mathbf{pk}, \mathbf{sk})$ denote the signer's public/private key pair, $\sigma_1$ denote a signature on block $m_1 \in \mathbb{Z}_p$, $\sigma_2$ denote a signature on block $m_2 \in \mathbb{Z}_p$.

- *Blockless verifiability:* Given $\sigma_1$ and $\sigma_2$, two random values $\alpha_1$, $\alpha_2 \in \mathbb{Z}_p$ and a block $m' = \alpha_1 m_1 + \alpha_2 m_2 \in \mathbb{Z}_p$, a verifier is able to check the correctness of block $m'$ without knowing block $m_1$ and $m_2$.
- *Non-malleability:* Given $\sigma_1$ and $\sigma_2$, two random values $\alpha_1$, $\alpha_2 \in \mathbb{Z}_p$ and a block $m' = \alpha_1 m_1 + \alpha_2 m_2 \in \mathbb{Z}_p$, a user, who does not have private key $\mathbf{sk}$, is not able to generate a valid signature $\sigma'$ on block $m'$ by linearly combining signature $\sigma_1$ and $\sigma_2$.

Blockless verifiability allows a verifier to audit the correctness of data stored in the cloud server with a special block, which is a linear combination of all the blocks in data. If the integrity of the combined block is correct, then the verifier believes that the integrity of the entire data is correct. In this way, the verifier does not need to download all the blocks to check the integrity of data. Non-malleability indicates that an adversary cannot generate valid signatures on arbitrary blocks by linearly combining existing signatures.

## 4 New Ring Signature Scheme

### 4.1 Overview

As we introduced in previous sections, we intend to utilize ring signatures to hide the identity of the signer on each block, so that private and sensitive information of the group is not disclosed to public verifiers. However, traditional ring signatures [21], [28] cannot be directly used into public auditing mechanisms, because these ring signature schemes do not support blockless verifiability. Without blockless verifiability, a public verifier has to download the whole data file to verify the correctness of shared data, which consumes excessive bandwidth and takes very long verification times.

Therefore, we design a new homomorphic authenticable ring signature (HARS) scheme, which is extended from a classic ring signature scheme [21]. The ring signatures generated by HARS are not only able to preserve identity privacy but also able to support blockless verifiability. We will show how to build the privacy-preserving public auditing mechanism for shared data in the cloud based on this new ring signature scheme in the next section.

### 4.2 Construction of HARS

HARS contains three algorithms: **KeyGen**, **RingSign** and **RingVerify**. In **KeyGen**, each user in the group generates his/her public key and private key. In **RingSign**, a user in the group is able to generate a signature on a block and its block identifier with his/her private key and all the group members' public keys. A block identifier is a string that can distinguish the corresponding block from others. A verifier is able to check whether a given block is signed by a group member in **RingVerify**. Details of this scheme are described in Fig. 3.

### 4.3 Security Analysis of HARS

Now, we discuss security properties of HARS, including correctness, unforgeability, blockless verifiability, non-malleability and identity privacy.

**Theorem 1.** *Given any block $m$, its block identifier $id$, and its ring signature $\sigma = (\sigma_1, \ldots, \sigma_{\mathbf{d}})$, a verifier is able to correctly check the integrity of this block under HARS.*

**Proof.** Based on properties of bilinear maps, the correctness of this scheme can be proved as follows:

Let $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$ be multiplicative cyclic groups of order $p$, $g_1$ and $g_2$ be generators of $\mathbb{G}_1$ and $\mathbb{G}_2$ respectively. Let $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ be a bilinear map, and $\psi : \mathbb{G}_2 \rightarrow \mathbb{G}_1$ be a computable isomorphism with $\psi(g_2) = g_1$. There is a public map-to-point hash function $H_1$: $\{0,1\}^* \rightarrow \mathbb{G}_1$, which can map a string $\{0,1\}^*$ into an element of $\mathbb{G}_1$ (i.e., an point on an elliptic curve). The total number of users in the group is $d$. The global parameters are $(e, \psi, p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, H_1, d)$.

**KeyGen.** For a user $u_i$, he/she randomly picks $x_i \overset{R}{\leftarrow} \mathbb{Z}_p$ and computes $w_i = g_2^{x_i} \in \mathbb{G}_2$. Then, user $u_i$'s public key is $\mathbf{pk}_i = w_i$ and his/her private key is $\mathbf{sk}_i = x_i$.

**RingSign.** Given all the $d$ users' public keys $(\mathbf{pk}_1, ..., \mathbf{pk}_d) = (w_1, ..., w_d)$, a block $m \in \mathbb{Z}_p$, the identifier of this block $id$ and the private key $\mathbf{sk}_s$ for some $s$, user $u_s$ randomly chooses $a_i \in \mathbb{Z}_p$ for all $i \neq s$, where $i \in [1, d]$, and let $\sigma_i = g_1^{a_i}$. Then, he/she computes

$$\beta = H_1(id) g_1^m \in \mathbb{G}_1, \tag{1}$$

and sets

$$\sigma_s = \left( \frac{\beta}{\psi(\prod_{i \neq s} w_i^{a_i})} \right)^{1/x_s} \in \mathbb{G}_1. \tag{2}$$

The ring signature of block $m$ is $\boldsymbol{\sigma} = (\sigma_1, ..., \sigma_d) \in \mathbb{G}_1^d$.

**RingVerify.** Given all the $d$ users' public keys $(\mathbf{pk}_1, ..., \mathbf{pk}_d) = (w_1, ..., w_d)$, a block $m$, an identifier $id$ and a ring signature $\boldsymbol{\sigma} = (\sigma_1, ..., \sigma_d)$, a verifier first computes $\beta = H_1(id) g_1^m \in \mathbb{G}_1$, and then checks

$$e(\beta, g_2) \overset{?}{=} \prod_{i=1}^{d} e(\sigma_i, w_i). \tag{3}$$

If the above equation holds, then the given block $m$ is signed by one of these $d$ users in the group. Otherwise, it is not.

Fig. 3. Details of HARS.

$$\begin{aligned}
\prod_{i=1}^{d} e(\sigma_i, w_i) &= e(\sigma_s, w_s) \cdot \prod_{i \neq s} e(\sigma_i, w_i) \\
&= e\left( \left( \frac{\beta}{\psi(\prod_{i \neq s} w_i^{a_i})} \right)^{\frac{1}{x_s}}, g_2^{x_s} \right) \cdot \prod_{i \neq s} e\left(g_1^{a_i}, g_2^{x_i}\right) \\
&= e\left( \frac{\beta}{\psi(\prod_{i \neq s} g_2^{x_i a_i})}, g_2 \right) \cdot \prod_{i \neq s} e\left(g_1^{a_i x_i}, g_2\right) \\
&= e\left( \frac{\beta}{\prod_{i \neq s} {g_1}^{a_i x_i}}, g_2 \right) \cdot e\left( \prod_{i \neq s} g_1^{a_i x_i}, g_2 \right) \\
&= e\left( \frac{\beta}{\prod_{i \neq s} {g_1}^{a_i x_i}} \cdot \prod_{i \neq s} {g_1}^{a_i x_i}, g_2 \right) \\
&= e(\beta, g_2),
\end{aligned}$$

where $\beta$ is computed as $\beta = H_1(id) g_1^m$. □

**Theorem 2.** *For any adversary $\mathcal{A}$, it is computationally infeasible to forge a ring signature under HARS, as long as the Co-CDH assumption on $(\mathbb{G}_1, \mathbb{G}_2)$ holds.*

**Proof.** We follow the security game defined in traditional ring signature schemes [21]. In the game, an adversary $\mathcal{A}$ is given all the $d$ users' public key $(\mathbf{pk}_1, \ldots, \mathbf{pk}_d) = (w_1, \ldots, w_d)$, and is given access to the hash oracle and the ring signing oracle. The goal of adversary $\mathcal{A}$ is to output a valid ring signature on a pair of block/identifier $(m, id)$, where this pair of block/identifier $(m, id)$ has never been presented to the ring signing oracle. If adversary $\mathcal{A}$ achieves this goal, then it wins the game. And we can prove that if adversary $\mathcal{A}$ could win the game, then we can find an algorithm $\mathcal{B}$ to solve the Co-CDH problem on $(\mathbb{G}_1, \mathbb{G}_2)$ with a non-negligible advantage, which contradicts to the Co-CDH assumption we introduced in Section 3. So, we first assume adversary $\mathcal{A}$ is able to generate a forgery by the following security game simulated by algorithm $\mathcal{B}$.

Initially, given $g_1^{ab} \in \mathbb{G}_1$, $g_2^a \in \mathbb{G}_2$, algorithm $\mathcal{B}$ randomly picks $x_2, \ldots, x_n$ from $\mathbb{Z}_p$ and sets $x_1 = 1$. Then, it sets $\mathbf{pk}_i = w_i = (g_2^a)^{x_i}$. Adversary $\mathcal{A}$ is given the public keys $(\mathbf{pk}_1, \ldots, \mathbf{pk}_d) = (w_1, \ldots, w_d)$. Without loss of generality, we assume $\mathcal{A}$ can submit distinct queries, which means for every ring signing query on a block $m$ and its identifier $id$, $\mathcal{A}$ has previously issued a hash query on block $m$ and identifier $id$.

*Hash Query.* On a hash query issued by $\mathcal{A}$, $\mathcal{B}$ flips a coin that shows 0 with probability $p_c$, and shows 1 otherwise. Then $\mathcal{B}$ randomly picks $r \in \mathbb{Z}_p$, if the coin shows 0, $\mathcal{B}$ returns $(g_1^{ab})^r$ to $\mathcal{A}$, otherwise it returns $\psi(g_2^a)^r$. Since $r$ is randomly selected from $\mathbb{Z}_p$, and $g_1^{ab}$ and $\psi(g_2^a)$ are both elements of cyclic group $\mathbb{G}_1$, therefore, the distribution of $(g_1^{ab})^r$ is identical to the distribution of $\psi(g_2^a)^r$, which means $\mathcal{A}$ cannot distinguish the result of flipped coin from the result of the hash query.

*Ring Signing Query.* Suppose $\mathcal{A}$ issues a ring signing query on a block $m$ and its identifier $id$. By the assumption of the game, a hash query has been issued by $\mathcal{B}$ on this pair of block/identifier $(m, id)$. If the coin $\mathcal{B}$ flipped for this hash query showed $0$, then $\mathcal{B}$ fails and exits. Otherwise $\mathcal{B}$ has returned $H(id) g_1^m = \psi(g_2^a)^r$ for some $r$, which was randomly selected in the corresponding hash query. In this case, $\mathcal{B}$ chooses random $a_2, \ldots, a_d \in \mathbb{Z}_p$, computes $a = r - (a_2 x_2 + \cdots + a_d x_d)$, and returns the signature $\boldsymbol{\sigma} = (g_1^a, g_1^{a_2}, \ldots, g_1^{a_d})$.

Eventually $\mathcal{A}$ outputs a forgery $\boldsymbol{\sigma} = (\boldsymbol{\sigma}_\mathbf{1}, \ldots, \boldsymbol{\sigma}_\mathbf{d})$ on block $m$ and identifier $id$. Again by the assumption, a hash query has been issued on this pair of block/identifier $(m, id)$. If the coin flipped by $\mathcal{B}$ for this hash query did not show $0$ then $\mathcal{B}$ fails. Otherwise, $H(id) g_1^m = g_1^{abr}$ for some $r$, which was randomly chosen by $\mathcal{B}$ in corresponding hash query, and $\mathcal{B}$ can output $g_1^b$ by computing $(\prod_{i=1}^{d} \sigma_i^{x_i})^{1/r}$.

Clearly, given $g_1^{ab} \in \mathbb{G}_1$, $g_2^a \in \mathbb{G}_2$, algorithm $\mathcal{B}$ is able to output $g_1^b \in \mathbb{G}_1$ via the above security game with an advantage of $\epsilon p_c^{q_s}(1 - p_c)$, if $\mathcal{A}$ is able to generate a forgery with an advantage of $\epsilon$. This advantage of algorithm $\mathcal{B}$ is maximized as $\epsilon / (e \cdot (1 + q_s))$ when $p_c = q_s / (q_s + 1)$, where $\mathbf{e} = \lim_{q_s \rightarrow \infty} (1 + 1/q_s)^{q_s}$.

According to [21], the Co-CDH problem can be solved by running two random instances of algorithm $\mathcal{B}$. Therefore, if adversary $\mathcal{A}$ is able to generate a forgery of a ring signature with an advantage of $\epsilon$, then running two instances of algorithm $\mathcal{B}$ is able to solve Co-CDH problem with an advantage of $(\epsilon/(\mathrm{e}+\mathrm{e}q_s))^2$. Clearly, if the advantage of adversary $\mathcal{A}$ of generating a forgery is non-negligible, then the advantage of solving the Co-CDH problem by running two instances of algorithm $\mathcal{B}$ is also non-negligible, which contradicts to the Co-CDH assumption on $(\mathbb{G}_1, \mathbb{G}_2)$. Therefore, it is computationally infeasible for adversary $\mathcal{A}$ to generate a forgery. □

**Theorem 3.** *HARS is a homomorphic authenticable ring signature scheme.*

**Proof.** To prove HARS is a homomorphic authenticable ring signature scheme, we first prove that HARS is able to support blockless verifiability, which we defined in Section 3. Then we show HARS is also non-malleable.

**Blockless Verifiability.** Given all the $d$ users' public keys $(\mathbf{pk}_1, \ldots, \mathbf{pk}_d) = (w_1, \ldots, w_d)$, two identifiers $id_1$ and $id_2$, two ring signatures $\boldsymbol{\sigma}_1 = (\boldsymbol{\sigma}_{1,1}, \ldots, \boldsymbol{\sigma}_{1,d})$ and $\sigma_{\mathbf{2}} = (\boldsymbol{\sigma}_{\mathbf{2},1}, \ldots, \boldsymbol{\sigma}_{\mathbf{2},\mathbf{d}})$, and two random values $y_1, y_2 \in \mathbb{Z}_p$, a verifier is able to check the correctness of a combined block $m' = y_1 m_1 + y_2 m_2 \in \mathbb{Z}_p$ without knowing block $m_1$ and $m_2$ by verifying:

$$e\big(H_1(id_1)^{y_1} H_1(id_2)^{y_2} g_1^{m'}, g_2\big) \stackrel{?}{=} \prod_{i=1}^{d} e\big(\sigma_{1,i}^{y_1} \cdot \sigma_{2,i}^{y_2}, w_i\big).$$

Based on Theorem 1, the correctness of the above equation can be proved as

$$\begin{aligned} & e(H_1(id_1)^{y_1} H_1(id_2)^{y_2} g_1^{m'}, g_2) \\ &\quad = e(H_1(id_1)^{y_1} g_1^{y_1 m_1}, g_2) \cdot e(H_1(id_2)^{y_2} g_1^{y_2 m_2}, g_2) \\ &\quad = e(\beta_1, g_2)^{y_1} \cdot e(\beta_2, g_2)^{y_2} \\ &\quad = \prod_{i=1}^{d} e(\sigma_{1,i}, w_i)^{y_1} \cdot \prod_{i=1}^{d} e(\sigma_{2,i}, w_i)^{y_2} \\ &\quad = \prod_{i=1}^{d} e\big(\sigma_{1,i}^{y_1} \cdot \sigma_{2,i}^{y_2}, w_i\big). \end{aligned}$$

If the combined block $m'$ is correct, the verifier also believes that block $m_1$ and $m_2$ are both correct. Therefore, HARS is able to support blockless verifiability.

*Non-Malleability.* Meanwhile, an adversary, who does not have any user's private key, cannot generate a valid ring signature $\boldsymbol{\sigma}'$ on the combined block $m' = y_1 m_1 + y_2 m_2$ by combining $\boldsymbol{\sigma}_\mathbf{1}$ and $\boldsymbol{\sigma}_2$ with $y_1$ and $y_2$. Because if an element $\boldsymbol{\sigma}'_i$ in $\boldsymbol{\sigma}'$ is computed as $\boldsymbol{\sigma}'_i = \boldsymbol{\sigma}_{1,i}^{y_1} \cdot \boldsymbol{\sigma}_{2,i}^{y_2}$, the whole ring signature $\boldsymbol{\sigma}' = (\boldsymbol{\sigma}'_1, \ldots, \boldsymbol{\sigma}'_d)$ cannot pass Equation (3) in *RingVerify*. The hardness of this problem lies in the fact that $H_1$ must be a one-way hash function (given every input, it is easy to compute; however, given the image of a random input, it is hard to invert).

Specifically, if block $m_1$ and $m_2$ are signed by the same user, for example, user $u_s$, then $\sigma'_s$ can be computed as

$$\sigma'_s = \sigma_{1,s}^{y_1} \cdot \sigma_{2,s}^{y_2} = \left(\frac{\beta_1^{y_1} \beta_2^{y_2}}{\prod_{i \neq s} w_{1,i}^{y_1 a_{1,i}} \cdot w_{2,i}^{y_2 a_{2,i}}}\right)^{1/x_s}.$$

For all $i \neq s$, $\sigma'_i = \sigma_{1,i}^{y_1} \cdot \sigma_{2,i}^{y_2} = g_1^{(y_1 a_{1,i} + y_2 a_{2,i})}$, where $a_{1,i}$ and $a_{2,i}$ are random values. When this invalid ring signature $\boldsymbol{\sigma}' = (\sigma'_1, \ldots, \sigma'_d)$ and the combined block $m'$ are verified together with Equation (3), we have

$$\prod_{i=1}^{d} e\big(\sigma'_i, w_i\big) = e\big(\beta_1^{y_1} \beta_2^{y_2}, g_2\big) \neq e(\beta', g_2),$$

which means it fails to pass the verification. The reason is that if $\beta_1^{y_1} \beta_2^{y_2} = H(id_1)^{y_1} H(id_2)^{y_2} g_1^{m'}$ is equal to $\beta' = H(id') g_1^{m'}$, we can have $H(id') = H(id_1)^{y_1} H(id_2)^{y_2}$. Then, given a random value of $h = H(id_1)^{y_1} H(id_2)^{y_2}$ (due to $y_1, y_2$ are randoms), we can easily find an input $id'$ so that $H(id') = h$, which contradicts to the assumption that $H_1$ is a one-way hash function.

If block $m_1$ and $m_2$ are signed by different users, for example, user $u_s$ and user $u_t$, then $\sigma'_s$ and $\sigma'_t$ can be presented as

$$\sigma'_s = \left(\frac{\beta_1^{y_1}}{\prod_{i \neq s} w_i^{y_1 a_{1,i}}}\right)^{1/x_s} \cdot g_1^{y_2 a_{2,s}},$$

$$\sigma'_t = g_1^{y_1 a_{1,t}} \cdot \left(\frac{\beta_2^{y_2}}{\prod_{i \neq t} w_i^{y_2 a_{2,i}}}\right)^{1/x_t}$$

For all $i \neq s$ and $i \neq t$, $\sigma'_i = \sigma_{1,i}^{y_1} \cdot \sigma_{2,i}^{y_2} = g_1^{(y_1 a_{1,i} + y_2 a_{2,i})}$, where $a_{1,i}$ and $a_{2,i}$ are random values. When this invalid ring signature $\sigma' = (\sigma'_\mathbf{1}, \ldots, \sigma'_\mathbf{d})$ and the combined block $m'$ are verified together with Equation (3), we have

$$\prod_{i=1}^{d} e\big(\sigma'_i, w_i\big) = e\big(\beta_1^{y_1} \beta_2^{y_2}, g_2\big) \neq e(\beta', g_2),$$

which means it fails to pass the verification, due to the same reason explained in the previous case that $H_1$ is a one-way hash function. Therefore, an adversary cannot output valid ring signatures on combined blocks by combining existing signatures, which indicates that HARS is non-malleable. Since HARS is blockless verifiable and non-malleable, it is a homomorphic authenticable signature scheme. □

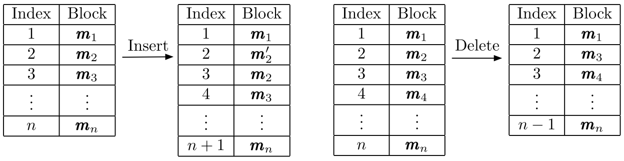
**Theorem 4.** *For any algorithm $\mathcal{A}$, any group $U$ with $d$ users, and a random user $u_s \in U$, the probability $Pr[\mathcal{A}(\boldsymbol{\sigma}) = u_s]$ is at most $1/d$ under HARS, where $\sigma$ is a ring signature generated with user $u_s$'s private key* $\mathbf{sk}_s$.

**Proof.** For any $h \in \mathbb{G}_1$, and any $s$, $1 \leq s \leq d$, the distribution $\{g_1^{a_1}, \ldots, g_1^{a_d} : a_i \xleftarrow{R} \mathbb{Z}_p$ for $i \neq s$, $a_s$ chosen such that $\prod_{i=1}^{d} g_1^{a_i} = h\}$ is identical to the distribution $\{g_1^{a_1}, \ldots, g_1^{a_d} : \prod_{i=1}^{d} g_1^{a_i} = h\}$. Therefore, given $\boldsymbol{\sigma} = (\sigma_1, \ldots, \sigma_d)$, the probability algorithm $\mathcal{A}$ distinguishes $\sigma_s$, which indicates the identity of the signer, is at most $1/d$. Further explanations of this proof about identity privacy can be found in [21]. □

## 5 Public Auditing Mechanism

### 5.1 Overview

Using HARS and its properties we established in the previous section, we now construct Oruta, a privacy-preserving public auditing mechanism for shared data in the cloud. With Oruta, the public verifier can verify the integrity of shared data without retrieving the entire data. Meanwhile,


Insert
Delete

| Index | Block |
|---|---|
| 1 | $\boldsymbol{m}_1$ |
| 2 | $\boldsymbol{m}_2$ |
| 3 | $\boldsymbol{m}_3$ |
| $\vdots$ | $\vdots$ |
| $n$ | $\boldsymbol{m}_n$ |

| Index | Block |
|---|---|
| 1 | $\boldsymbol{m}_1$ |
| 2 | $\boldsymbol{m}_2'$ |
| 3 | $\boldsymbol{m}_2$ |
| 4 | $\boldsymbol{m}_3$ |
| $\vdots$ | $\vdots$ |
| $n+1$ | $\boldsymbol{m}_n$ |

(a) After inserting block $\boldsymbol{m}_2'$, all the identifiers after block $\boldsymbol{m}_2'$ are changed

| Index | Block |
|---|---|
| 1 | $\boldsymbol{m}_1$ |
| 2 | $\boldsymbol{m}_2$ |
| 3 | $\boldsymbol{m}_3$ |
| 4 | $\boldsymbol{m}_4$ |
| $\vdots$ | $\vdots$ |
| $n$ | $\boldsymbol{m}_n$ |

| Index | Block |
|---|---|
| 1 | $\boldsymbol{m}_1$ |
| 2 | $\boldsymbol{m}_3$ |
| 3 | $\boldsymbol{m}_4$ |
| $\vdots$ | $\vdots$ |
| $n-1$ | $\boldsymbol{m}_n$ |

(b) After deleting block $\boldsymbol{m}_2$, all the identifiers after block $\boldsymbol{m}_1$ are changed

Fig. 4. Using indices as identifiers.

the identity of the signer on each block in shared data is kept private from the public verifier during the auditing.

### 5.2 Reduce Signature Storage

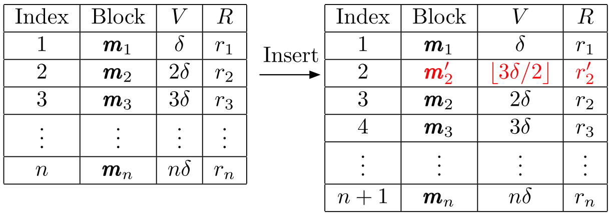
Another important issue we should consider in the construction of Oruta is the size of storage used for ring signatures. According to the generation of ring signatures in HARS, a block $m$ is an element of $\mathbb{Z}_p$ and its ring signature contains $d$ elements of $\mathbb{G}_1$, where $\mathbb{G}_1$ is a cyclic group with order $p$. It means a $|p|$-bit block requires a $d \times |p|$-bit ring signature, which forces users to spend a huge amount of space on storing ring signatures. It will be very frustrating for users, because cloud service providers, such as Amazon, will charge users based on the storage space they use.

To reduce the storage of ring signatures on shared data and still allow the public verifier to audit shared data efficiently, we exploit an aggregated approach from [10] to expand the size of each block in shared data into $k \times |p|$ bits. Specifically, a block in shared data is denoted as $\boldsymbol{m}_j = (m_{j,1}, \dots, m_{j,k}) \in \mathbb{Z}_p^k$, for $1 \leq j \leq n$ and $n$ is the total number of blocks. To generate a ring signature on block $\boldsymbol{m}_j$ with HARS, a user aggregates block $m_j = (m_{j,1}, \dots, m_{j,k})$ as $\prod_{l=1}^{k} \eta_l^{m_{j,l}}$ instead of computing $g_1^m$ in Equation (1), where $\eta_1, \dots, \eta_k$ are random values of $\mathbb{G}_1$. With the aggregation of a block, the length of a ring signature is only $d/k$ of the length of a block. Similar methods to reduce the storage space of signatures can also be found in [15]. Generally, to obtain a smaller size of a ring signature than the size of a block, we choose $k > d$. As a trade-off, the communication cost of an auditing task will be increasing with an increase of $k$ (analyzed in later section).

### 5.3 Support Dynamic Operations

To enable each user in the group to easily modify data in the cloud, Oruta should also support dynamic operations on shared data. A dynamic operation includes an insert, delete or update operation on a single block [9]. However, since the computation of a ring signature includes an identifier of a block (as presented in HARS), traditional methods, which only use the index of a block as its identifier (i.e., the index of block $\boldsymbol{m}_j$ is $j$), are not suitable for supporting dynamic operations on shared data efficiently.

The reason is that, when a user modifies a single block in shared data by performing an insert or delete operation, the indices of blocks that after the modified block are all changed (as shown in Fig. 4), and the changes of these indices require users, who are sharing the data, to re-compute the signatures of these blocks, even though the content of these blocks are not modified.



| Index | Block | $V$ | $R$ |
|---|---|---|---|
| 1 | $\boldsymbol{m}_1$ | $\delta$ | $r_1$ |
| 2 | $\boldsymbol{m}_2$ | $2\delta$ | $r_2$ |
| 3 | $\boldsymbol{m}_3$ | $3\delta$ | $r_3$ |
| $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ |
| $n$ | $\boldsymbol{m}_n$ | $n\delta$ | $r_n$ |

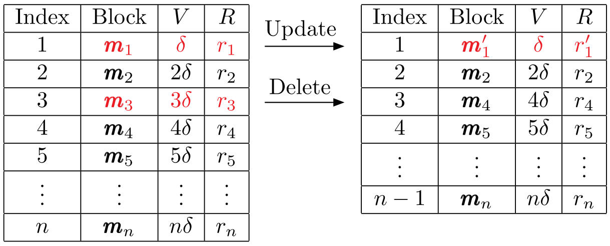
| Index | Block | $V$ | $R$ |
|---|---|---|---|
| 1 | $\boldsymbol{m}_1$ | $\delta$ | $r_1$ |
| 2 | $\boldsymbol{m}_2'$ | $\lfloor 3\delta/2 \rfloor$ | $r_2'$ |
| 3 | $\boldsymbol{m}_2$ | $2\delta$ | $r_2$ |
| 4 | $\boldsymbol{m}_3$ | $3\delta$ | $r_3$ |
| $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ |
| $n+1$ | $\boldsymbol{m}_n$ | $n\delta$ | $r_n$ |

Fig. 5. Insert block $\mathbf{m}_2'$ into shared data using an index hash table as identifiers.

By utilizing an index hash table [15], which is a data structure indexing each block based on its hash value, our mechanism can allow a user to efficiently perform a dynamic operation on a single block, and avoid this type of re-computation on other blocks. Different from [15], in our mechanism, an identifier from the index hash table is described as $id_j = \{v_j, r_j\}$, where $v_j$ is denoted as the virtual index of block $\boldsymbol{m}_j$, and $r_j$ is a random generated by a collision-resistant hash function $H_2 : \{0,1\}^* \rightarrow \mathbb{Z}_q$ with $r_j = H_2(\boldsymbol{m}_j \| v_j)$. Here, $q$ is a much smaller prime than $p$ (e.g., $|q| = 80$ bits and $|p| = 160$ bits). Examples of different dynamic operations on shared data with our index hash tables are described in Figs. 5 and 6.

Specifically, the value of $r$ generated by $H_2$ ensures that each block has a unique identifier (i.e., the probability that two blocks have the same value of $r$ is negligible). The virtual indices are able to ensure that all the blocks in shared data are in the right order. For example, if $v_i < v_j$, then block $\boldsymbol{m}_i$ is ahead of block $\boldsymbol{m}_j$ in shared data. When shared data is created by the original user, the initial virtual index of block $\mathbf{m_j}$ is computed as $v_j = j \cdot \delta$, where $\delta$ is a system parameter decided by the original user. If a new block $\boldsymbol{m}_j'$ is inserted, the virtual index of this new block $\boldsymbol{m}_j'$ is $v_j' = \lfloor (v_{j-1} + v_j)/2 \rfloor$. Clearly, if block $\boldsymbol{m}_j$ and block $\boldsymbol{m}_{j+1}$ are both originally created by the original user, the maximum number of inserted blocks that is allowed between block $\boldsymbol{m}_j$ and block $\boldsymbol{m}_{j+1}$ is $\delta$. In this paper, we assume the value of $\delta$ is always large enough to support a sufficient number of insertions between two blocks for the group, so that any two virtual indexes in the table will not be the same.

To support dynamic data without the above assumption, the combination of coding techniques and Oblivious RAM can be utilized as introduced in recent work [29]. Unfortunately, this proposed solution in [29] requires much more computation and communication overhead. Moreover, the property of public verifiability is inevitably sacrificed in [29].


Update
Delete

| Index | Block | $V$ | $R$ |
|---|---|---|---|
| 1 | $\boldsymbol{m}_1$ | $\delta$ | $r_1$ |
| 2 | $\boldsymbol{m}_2$ | $2\delta$ | $r_2$ |
| 3 | $\boldsymbol{m}_3$ | $3\delta$ | $r_3$ |
| 4 | $\boldsymbol{m}_4$ | $4\delta$ | $r_4$ |
| 5 | $\boldsymbol{m}_5$ | $5\delta$ | $r_5$ |
| $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ |
| $n$ | $\boldsymbol{m}_n$ | $n\delta$ | $r_n$ |

| Index | Block | $V$ | $R$ |
|---|---|---|---|
| 1 | $\boldsymbol{m}_1'$ | $\delta$ | $r_1'$ |
| 2 | $\boldsymbol{m}_2$ | $2\delta$ | $r_2$ |
| 3 | $\boldsymbol{m}_4$ | $4\delta$ | $r_4$ |
| 4 | $\boldsymbol{m}_5$ | $5\delta$ | $r_5$ |
| $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ |
| $n-1$ | $\boldsymbol{m}_n$ | $n\delta$ | $r_n$ |

Fig. 6. Update block $\boldsymbol{m}_1$ and delete block $\boldsymbol{m}_3$ in shared data using an index hash table as identifiers.

### 5.4 Construction of Oruta

Now, we present the details of our public auditing mechanism. It includes five algorithms: **KeyGen**, **SigGen**, **Modify**, **ProofGen** and **ProofVerify**. In **KeyGen**, users generate their own public/private key pairs. In **SigGen**, a user (either the original user or a group user) is able to compute ring signatures on blocks in shared data by using its own private key and all the group members' public keys. Each user in the group is able to perform an insert, delete or update operation on a block, and compute the new ring signature on this new block in **Modify**. **ProofGen** is operated by a public verifier and the cloud server together to interactively generate a proof of possession of shared data. In **ProofVerify**, the public verifier audits the integrity of shared data by verifying the proof.

Note that for the ease of understanding, we first assume the group is static, which means the group is pre-defined before shared data is created in the cloud and the membership of the group is not changed during data sharing. Specifically, before the original user outsources shared data to the cloud, he/she decides all the group members. We will discuss the case of dynamic groups later.

*Discussion.* In the construction of Oruta, we support data privacy by leveraging random masking (i.e., $\tau_l h(\lambda_l)$ in **ProofGen**), which is also used in previous work [5] to protect data privacy for personal users. If a user wants to protect the content of private data in the cloud, this user can also encrypt data before outsourcing it into the cloud server with encryption techniques [30], [31], such as the combination of symmetric key encryption and attribute-based encryption (ABE) [30].

With the sampling strategy [9], which is widely used in most of the public auditing mechanisms, a public verifier can detect any corrupted block in shared data with a high probability by only choosing a subset of all blocks (i.e., choosing $c$-element subset $\mathcal{J}$ from set $[1, n]$) in each auditing task. Previous work [9] has already proved that, given a total number of blocks $n = 1{,}000{,}000$, if $1$ percent of all the blocks are lost or removed, a public verifier can detect these corrupted blocks with a probability greater than $99$ percent by choosing only $460$ random blocks. Of course, this public verifier can always spend more communication overhead, and verify the integrity of data by choosing all the $n$ blocks in shared data. Even if all the $n$ blocks in shared data are selected (i.e., without using sampling strategy), the communication overhead during public auditing is still much more smaller than retrieving the entire data from the cloud [9].

Besides choosing a larger number of random blocks, another possible approach to improve the detection probability is to perform multiple auditing tasks on the same shared data by using different randoms (i.e., $y_j$ is different for block $\mathbf{m_j}$ in each different task). Specifically, if the current detection probability is $P_x$ and a number of $t$ auditing tasks is performed, then the detection probability is computed as $1 - (1 - P_x)^t$.

*Dynamic Groups.* We now discuss the scenario of dynamic groups under our proposed mechanism. If a new user can be added in the group or an existing user can be revoked from the group, then this group is denoted as a dynamic group. To support dynamic groups while still allowing the public verifier to perform public auditing, all the ring signatures on shared data need to be re-computed with the signer's private key and all the current users' public keys when the membership of the group is changed.

For example, if the current size of the group is $d$ and a new user $u_{d+1}$ is added into the group, then a ring signature on each block in shared data needs to be re-computed with the signer's private key and all the $d+1$ public keys $(\mathbf{pk}_1, \ldots, \mathbf{pk}_{d+1})$. If the current size of the group is $d$ and an existing user $u_d$ is revoked from the group, then a ring signature on each block in shared data needs to be re-computed with the signer's private key and all the $d-1$ public keys $(\mathbf{pk}_1, \ldots, \mathbf{pk}_{d-1})$.

The main reason of this type of re-computation on signatures introduced by dynamic groups, is because the generation of a ring signature under our mechanism requires the signer's private key and all the current members' public keys. An interesting problem for our future work will be how to avoid this type of re-computation introduced by dynamic groups while still preserving identity privacy from the public verifier during the process of public auditing on shared data.

### 5.5 Security Analysis of Oruta

Now, we discuss security properties of Oruta, including its correctness, unforgeability, identity privacy and data privacy.

**Theorem 5.** *A public verifier is able to correctly audit the integrity of shared data under Oruta.*

**Proof.** According to the description of **ProofVerify**, a public verifier believes the integrity of shared data is correct if Equation (6) holds. So, the correctness of our scheme can be proved by verifying the correctness of Equation (6). Based on properties of bilinear maps and Theorem 1, the right-hand side (RHS) of Equation (6) can be expanded as follows:

$$\begin{aligned}
\mathtt{RHS} &= \left(\prod_{i=1}^{d} e\left(\prod_{j\in\mathcal{J}} \sigma_{j,i}^{y_j}, w_i\right)\right) \cdot e\left(\prod_{l=1}^{k} \lambda_l^{h(\lambda_l)}, g_2\right) \\
&= \left(\prod_{j\in\mathcal{J}}\left(\prod_{i=1}^{d} e(\sigma_{j,i}, w_i)^{y_j}\right)\right) \cdot e\left(\prod_{l=1}^{k} \eta_l^{\tau_l h(\lambda_l)}, g_2\right) \\
&= \left(\prod_{j\in\mathcal{J}} e(\beta_j, g_2)^{y_j}\right) \cdot e\left(\prod_{l=1}^{k} \eta_l^{\tau_l h(\lambda_l)}, g_2\right) \\
&= e\left(\prod_{j\in\mathcal{J}}\left(H_1(id_j)\prod_{l=1}^{k}\eta_l^{m_{j,l}}\right)^{y_j}, g_2\right) \cdot e\left(\prod_{l=1}^{k} \eta_l^{\tau_l h(\lambda_l)}, g_2\right) \\
&= e\left(\prod_{j\in\mathcal{J}} H_1(id_j)^{y_j} \cdot \prod_{l=1}^{k} \eta_l^{\sum_{j\in\mathcal{J}} m_{j,l} y_j} \cdot \prod_{l=1}^{k} \eta_l^{\tau_l h(\lambda_l)}, g_2\right) \\
&= e\left(\prod_{j\in\mathcal{J}} H_1(id_j)^{y_j} \cdot \prod_{l=1}^{k} \eta_l^{\mu_l}, g_2\right).
\end{aligned}$$

□

**Theorem 6.** *For an untrusted cloud, it is computationally infeasible to generate a forgery of an auditing proof under Oruta as long as the DL assumption holds.*

Let $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$ be multiplicative cyclic groups of order $p$, $g_1$ and $g_2$ be generators of groups $\mathbb{G}_1$, $\mathbb{G}_2$, 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 total number of users in the group is $d$. Shared data $M$ is divided into $n$ blocks as $M = (\boldsymbol{m}_1, ..., \boldsymbol{m}_n)$, and each block $\boldsymbol{m}_j = (m_{j,1}, ..., m_{j,k})$ is further divided into $k$ elements of $\mathbb{Z}_p$. The global parameters are $(e, \psi, p, q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, H_1, H_2, h, d, n, k)$.

**KeyGen.** For user $u_i$, he/she randomly picks $x_i \in \mathbb{Z}_p$ and computes $w_i = g_2^{x_i}$. User $u_i$'s public key is $\mathbf{pk}_i = w_i$ and his/her private key is $\mathbf{sk}_i = x_i$. The original user also randomly generates a public aggregate key $\mathbf{pak} = (\eta_1, ..., \eta_k)$, where $\eta_l$ are random elements of $\mathbb{G}_1$.

**SigGen.** Given all the $d$ group members' public keys $(\mathbf{pk}_1, ..., \mathbf{pk}_d) = (w_1, ..., w_d)$, a block $\boldsymbol{m}_j = (m_{j,1}, ..., m_{j,k})$, its identifier $id_j$, a private key $\mathbf{sk}_s$ for some $s$, user $u_s$ computes a ring signature of this block as follows:

1) Aggregates block $\boldsymbol{m}_j$ with the public aggregate key $\mathbf{pak}$, and computes

$$\beta_j = H_1(id_j)\prod_{l=1}^{k}\eta_l^{m_{j,l}} \in \mathbb{G}_1. \tag{4}$$

2) Randomly chooses $a_{j,i} \in \mathbb{Z}_p$ and sets $\sigma_{j,i} = g_1^{a_{j,i}}$, for all $i \neq s$. Then, calculates

$$\sigma_{j,s} = \left(\frac{\beta_j}{\psi(\prod_{i\neq s} w_i^{a_{j,i}})}\right)^{1/x_s} \in \mathbb{G}_1. \tag{5}$$

The ring signature of block $\boldsymbol{m}_j$ is $\boldsymbol{\sigma}_j = (\sigma_{j,1}, ..., \sigma_{j,d})$.

**Modify.** A user in the group modifies the $j$-th block in shared data by performing one of the following three operations:

- **Insert.** This user inserts a new block $\boldsymbol{m}'_j$ into shared data. He/She computes the new identifier of the inserted block $\boldsymbol{m}'_j$ as $id'_j = \{v'_j, r'_j\}$, where virtual index $v'_j = \lfloor(v_{j-1} + v_j)/2\rfloor$, and $r'_j = H_2(\boldsymbol{m}'_j||v'_j)$. This user outputs the new ring signature $\boldsymbol{\sigma}'_j$ of the inserted block $\boldsymbol{m}'_j$ with **SigGen**, and uploads $\{\boldsymbol{m}'_j, id'_j, \boldsymbol{\sigma}'_j\}$ to the cloud server. For the rest of blocks, the identifiers of these blocks are not changed. The total number of blocks in shared data increases to $n + 1$.
- **Delete.** This user deletes block $\boldsymbol{m}_j$, its identifier $id_j$ and ring signature $\boldsymbol{\sigma}_j$ from the cloud server. The identifiers and content of other blocks in shared data are remain the same. The total number of blocks in shared data decreases to $n - 1$.
- **Update.** This user updates the $j$-th block in shared data with a new block $\boldsymbol{m}'_j$. The virtual index of this block is remain the same, and $r'_j$ is computed as $r'_j = H_2(\boldsymbol{m}'_j||v_j)$. The new identifier of this updated block is $id'_j = \{v_j, r'_j\}$. The identifiers of other blocks in shared data are not changed. This user outputs the new ring signature $\boldsymbol{\sigma}'_j$ of this new block with **SigGen**, and uploads $\{\boldsymbol{m}'_j, id'_j, \boldsymbol{\sigma}'_j\}$ to the cloud server. The total number of blocks in shared data is still $n$.

**ProofGen.** To audit the integrity of shared data, a public verifier:

1) Randomly picks a $c$-element subset $\mathcal{J}$ of set $[1, n]$ to locate the $c$ selected blocks that will be checked, where $n$ is total number of blocks in shared data.
2) Generates a random value $y_j \in \mathbb{Z}_q$, for $j \in \mathcal{J}$.
3) Sends an auditing challenge $\{(j, y_j)\}_{j\in\mathcal{J}}$ to the cloud server.

After receiving an auditing challenge $\{(j, y_j)\}_{j\in\mathcal{J}}$, the cloud server generates a proof of possession of selected blocks. More specifically, the cloud server:

1) Chooses a random element $\tau_l \in \mathbb{Z}_q$, and calculates $\lambda_l = \eta_l^{\tau_l} \in \mathbb{G}_1$, for $l \in [1, k]$.
2) Computes $\mu_l = \sum_{j\in\mathcal{J}} y_j m_{j,l} + \tau_l h(\lambda_l) \in \mathbb{Z}_p$, for $l \in [1, k]$.
3) Aggregates signatures as $\phi_i = \prod_{j\in\mathcal{J}} \sigma_{j,i}^{y_j}$, for $i \in [1, d]$.
4) Returns an auditing proof $\{\boldsymbol{\lambda}, \boldsymbol{\mu}, \boldsymbol{\phi}, \{id_j\}_{j\in\mathcal{J}}\}$ to the public verifier, where $\boldsymbol{\lambda} = (\lambda_1, ..., \lambda_k)$, $\boldsymbol{\mu} = (\mu_1, ..., \mu_k)$ and $\boldsymbol{\phi} = (\phi_1, ..., \phi_d)$.

**ProofVerify.** With an auditing proof $\{\boldsymbol{\lambda}, \boldsymbol{\mu}, \boldsymbol{\phi}, \{id_j\}_{j\in\mathcal{J}}\}$, an auditing challenge $\{(j, y_j)\}_{j\in\mathcal{J}}$, public aggregate key $\mathbf{pak} = (\eta_1, ..., \eta_k)$, and all the group members' public keys $(\mathbf{pk}_1, ..., \mathbf{pk}_d) = (w_1, ..., w_d)$, the public verifier checks the correctness of this proof by checking the following equation:

$$e(\prod_{j\in\mathcal{J}} H_1(id_j)^{y_j} \prod_{l=1}^{k} \eta_l^{\mu_l}, g_2) \stackrel{?}{=} \left(\prod_{i=1}^{d} e(\phi_i, w_i)\right) \cdot e(\prod_{l=1}^{k} \lambda_l^{h(\lambda_l)}, g_2). \tag{6}$$

If the above equation holds, then the public verifier believes that the blocks in shared data are all correct. Otherwise, the integrity of shared data is incorrect.

Fig. 7. Details of Oruta.

**Proof.** As proved in Theorem 2, for an untrusted cloud, if the Co-CDH problem on $(\mathbb{G}_1, \mathbb{G}_2)$ is hard, it is computationally infeasible to generate a forgery of a ring signature under HARS. In Oruta, besides trying to compute a forgery of a ring signature on each block to generate a forgery of an auditing proof, if the untrusted cloud could win the following security game, denoted as *Game 1*, it can generate a forgery of an auditing proof for corrupted shared data. Following the security game in [10], we describe this security game as follows:

*Game 1*. A public verifier sends an auditing challenge $\{(j, y_j)\}_{j\in\mathcal{J}}$ to the cloud server, the auditing proof generated based on the correct shared data $M$ should be $\{\boldsymbol{\lambda}, \boldsymbol{\mu}, \boldsymbol{\phi}, \{id_j\}_{j\in\mathcal{J}}\}$, which is able to pass the verification with Equation (6). The untrusted cloud generates a forgery of the proof as $\{\boldsymbol{\lambda}, \boldsymbol{\mu}', \boldsymbol{\phi}, \{id_j\}_{j\in\mathcal{J}}\}$ based on the corrupted shared data $M'$, where $\boldsymbol{\mu}' = (\mu'_1, \ldots, \mu'_k)$ and $\mu'_l = \sum_{j\in\mathcal{J}} y_j m'_{j,l} + \tau_l h(\lambda_l)$. Define $\Delta\mu_l = \mu'_l - \mu_l$ for $1 \leq l \leq k$, and at least one element of $\{\Delta\mu_l\}_{1\leq l\leq k}$ is nonzero since $M \neq M'$. If this invalid proof based on the corrupted shared data $M'$ can successfully pass the verification, then the untrusted cloud wins. Otherwise, it fails.

Now, we prove that, if the untrusted cloud could win Game 1, we can find a solution of solving the DL problem in $\mathbb{G}_1$, which contradicts to the DL assumption that the DL problem in $\mathbb{G}_1$ is computationally infeasible. We first assume the untrusted cloud can win Game 1. Then, according to Equation (6), we have

$$e\left(\prod_{j\in\mathcal{J}} H_1(id_j)^{y_j} \cdot \prod_{l=1}^{k} \eta_l^{\mu'_l}, g_2\right) = \left(\prod_{i=1}^{d} e(\phi_i, w_i)\right) e\left(\prod_{l=1}^{k} \lambda_l^{h(\lambda_l)}, g_2\right).$$

Because $\{\boldsymbol{\lambda}, \boldsymbol{\mu}, \boldsymbol{\phi}, \{id_j\}_{j\in\mathcal{J}}\}$ is a correct auditing proof, we also have

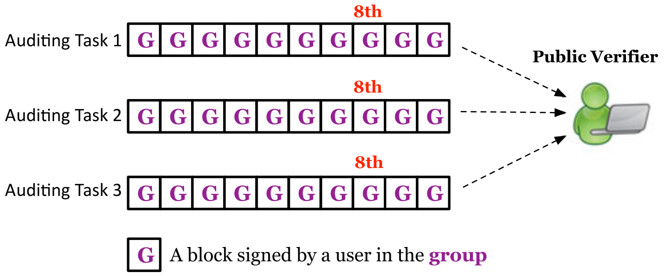


Fig. 8. Alice and Bob share a file in the cloud, and a public verifier audits the integrity of shared data with Oruta.

$$e\left(\prod_{j\in\mathcal{J}} H_1(id_j)^{y_j}\cdot\prod_{l=1}^{k}\eta_l^{\mu_l}, g_2\right) = \left(\prod_{i=1}^{d} e(\phi_i,w_i)\right) e\left(\prod_{l=1}^{k}\lambda_l^{h(\lambda_l)}, g_2\right).$$

Then, we can learn that

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

For two random elements $g, h \in \mathbb{G}_1$, there exists $x \in \mathbb{Z}_p$ that $h = g^x$ because $\mathbb{G}_1$ is a cyclic group. Without loss of generality, given $g$, $h \in \mathbb{G}_1$, each $\eta_l$ is able to randomly and correctly generated by computing $\eta_l = g^{\xi_l}h^{\gamma_l} \in \mathbb{G}_1$, where $\xi_l$ and $\gamma_l$ are random values of $\mathbb{Z}_p$. Then, we have

$$1 = \prod_{l=1}^{k}\eta_l^{\Delta\mu_l} = \prod_{l=1}^{k}(g^{\xi_l}h^{\gamma_l})^{\Delta\mu_l} = g^{\sum_{l=1}^{k}\xi_l\Delta\mu_l}\cdot h^{\sum_{l=1}^{k}\gamma_l\Delta\mu_l}.$$

Clearly, we can find a solution to the DL problem. More specifically, given $g$, $h^x \in \mathbb{G}_1$, we can compute

$$h = g^{\frac{\sum_{l=1}^{k}\xi_l\Delta\mu_l}{\sum_{l=1}^{k}\gamma_l\Delta\mu_l}} = g^x, \quad x = \frac{\sum_{l=1}^{k}\xi_l\Delta\mu_l}{\sum_{l=1}^{k}\gamma_l\Delta\mu_l}.$$

unless the denominator $\sum_{l=1}^{k}\gamma_l\Delta\mu_l$ 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 $\gamma_l$ is random element of $\mathbb{Z}_p$, therefore, the denominator is zero with probability of $1/p$. It means, if the untrusted cloud wins Game 1, we can find a solution to the DL problem with a probability of $1 - 1/p$, which is non-negligible. It contradicts the assumption that the DL problem is hard in $\mathbb{G}_1$. Therefore, for an untrusted cloud, it is computationally infeasible to generate a forgery of an auditing proof. □

Now, we show that the identity of the signer on each block in share data is not disclosed to a public verifier.

**Theorem 7.** *During public auditing, the probability for a public verifier to distinguish the identities of all the signers on the $c$ selected blocks in shared data is at most $1/d^c$.*

**Proof.** According to Theorem 4, for any algorithm $\mathcal{A}$, the probability to reveal the signer on one block is $1/d$. Because the $c$ selected blocks are signed independently, where $c \in [1, n]$, the total probability that the public verifier can distinguish all the signers' identities on the $c$ selected blocks in shared data is at most $1/d^c$. □

Assume the integrity of $B$ auditing tasks need to be verified, where the $B$ corresponding shared data are denoted as $M_1, ..., M_B$, the number of users sharing data $M_b$ is described as $d_b$, where $1 \le b \le B$.

**BatchProofGen.** A public verifier first generates an auditing challenge $\{(j, y_j)\}_{j\in\mathcal{J}}$ as in **ProofGen**. After receiving the auditing challenge, the cloud server generates and returns an auditing proof $\{\boldsymbol{\lambda}_b, \boldsymbol{\mu}_b, \boldsymbol{\phi}_b, \{id_{b,j}\}_{j\in\mathcal{J}}\}$ for each shared data $M_b$ as in **ProofGen**, where $1 \le b \le B$, $1 \le l \le k$, $1 \le i \le d_b$ and

$$\begin{cases} \lambda_{b,l} = \eta_{b,l}^{\tau_{b,l}} \\ \mu_{b,l} = \sum_{j\in\mathcal{J}} y_j m_{b,j,l} + \tau_{b,l}h(\lambda_{b,l}) \\ \phi_{b,i} = \sum_{j\in\mathcal{J}} \sigma_{b,j,i}^{y_j} \end{cases}$$

Here $id_{b,j}$ is described as $id_{b,j} = \{f_b, v_{b,j}, r_{b,j}\}$, where $f_b$ is the data identifier (e.g., file name) of shared data $M_b$.

**BatchProofVerify.** After receiving all the $B$ auditing proofs, the public verifier checks the correctness of these $B$ proofs simultaneously by checking the following equation with all the $\sum_{b=1}^{B} d_b$ users' public keys:

$$e\left(\prod_{b=1}^{B}\left(\prod_{j\in\mathcal{J}} H(id_{b,j})^{y_j}\cdot\prod_{l=1}^{k}\eta_{b,l}^{\mu_{b,l}}\right), g_2\right) \overset{?}{=} \left(\prod_{b=1}^{B}\prod_{i=1}^{d_b} e(\phi_{b,i}, w_{b,i})\right)\cdot e\left(\prod_{b=1}^{B}\prod_{l=1}^{k}\lambda_{b,l}^{h(\lambda_{b,l})}, g_2\right), \quad (7)$$

where $\mathbf{pk}_{b,i} = w_{b,i} = g^{x_{b,i}}$ and $\mathbf{sk}_{b,i} = x_{b,i}$. If the above verification equation holds, then the public verifier believes that the integrity of all the $B$ shared data is correct. Otherwise, there is at least one shared data is corrupted.

Fig. 9. Details of Batch Auditing.

Let us reconsider the example we described in Section 1. With our proposed mechanism, the public verifier knows each block in shared data is either signed by Alice or Bob, because it needs both users' public keys to verify the correctness of the entire shared data. However, it cannot distinguish who is the signer on each particular block (as shown in Fig. 8). Therefore, the public verifier cannot have additional advantages on revealing private information, such as who always signs the largest number of blocks in shared data or which particular block is frequently modified by different group members.

Following the similar theorem in [5], we show that our scheme is also able to support data privacy.

**Theorem 8.** Given an auditing proof $= \{\boldsymbol{\lambda}, \boldsymbol{\mu}, \boldsymbol{\phi}, \{id_j\}_{j\in\mathcal{J}}\}$, it is computationally infeasible for a public verifier to reveal any private data in shared data under Oruta as long as the DL assumption holds.

**Proof.** If the combined element $\sum_{j\in\mathcal{J}} y_j m_{j,l}$, which is a linear combination of all the elements in block $\mathbf{m_j}$, is directly sent to a public verifier, the public verifier could learn the content of data by solving linear equations after collecting a sufficient number of linear combinations. To preserve private data, the combined element is computed with random masking as $\mu_l = \sum_{j\in\mathcal{J}} y_j m_{j,l} + \tau_l h(\lambda_l)$. In order to still solve linear equations, the public verifier must know the value of $\tau_l \in \mathbb{Z}_p$. However, given $\eta_l \in \mathbb{G}_1$ $\lambda_l = \eta_l^{\tau_l} \in \mathbb{G}_1$, computing $\tau_l$ is as hard as solving the DL problem in $\mathbb{G}_1$, which is computationally infeasible. Therefore, give $\boldsymbol{\lambda}$ and $\boldsymbol{\mu}$, the public verifier cannot directly obtain any linear combination of all the elements in a block, and cannot further reveal any private data in shared data $M$. □

### 5.6 Batch Auditing

Sometimes, a public verifier may need to verify the correctness of multiple auditing tasks in a very short time. Directly verifying these multiple auditing tasks separately would be inefficient. By leveraging the properties of bilinear maps, we can further extend Oruta to support batch auditing, which can verify the correctness of multiple auditing tasks simultaneously and improve the efficiency of public auditing. Details of batch auditing are presented in Fig. 9.

Based on the correctness of Equation (6), the correctness of batch auditing in Equation (7) can be presented as

$$\begin{aligned}
&\left(\prod_{b=1}^{B}\prod_{i=1}^{d_b} e(\boldsymbol{\phi}_{b,i}, w_{b,i})\right)\cdot e\left(\prod_{b=1}^{B}\prod_{l=1}^{k}\lambda_{b,l}^{h(\lambda_{b,l})}, g_2\right)\\
&\quad=\prod_{b=1}^{B}\left(\left(\prod_{i=1}^{d_b} e(\boldsymbol{\phi}_{b,i}, w_{b,i})\right)\cdot e\left(\prod_{l=1}^{k}\lambda_{b,l}^{h(\lambda_{b,l})}, g_2\right)\right)\\
&\quad=\prod_{b=1}^{B} e\left(\prod_{j\in\mathcal{J}} H(id_{b,j})^{y_j}\cdot\prod_{l=1}^{k}\eta_{b,l}^{\mu_{b,l}}, g_2\right)\\
&\quad=e\left(\prod_{b=1}^{B}\left(\prod_{j\in\mathcal{J}} H(id_{b,j})^{y_j}\cdot\prod_{l=1}^{k}\eta_{b,l}^{\mu_{b,l}}\right), g_2\right).
\end{aligned}$$

If all the $B$ shared data are from the same group, the public verifier can further improve the efficiency of batch auditing by verifying

$$\begin{aligned}
&e\left(\prod_{b=1}^{B}\left(\prod_{j\in\mathcal{J}} H(id_{b,j})^{y_j}\cdot\prod_{l=1}^{k}\eta_{b,l}^{\mu_{b,l}}\right), g_2\right)\\
&\quad\overset{?}{=}\left(\prod_{i=1}^{d} e(\prod_{b=1}^{B}\boldsymbol{\phi}_{b,i}, w_i)\right)\cdot e\left(\prod_{b=1}^{B}\prod_{l=1}^{k}\lambda_{b,l}^{h(\lambda_{b,l})}, g_2\right),
\end{aligned}\tag{8}$$

which can save the public verifier about $(d-1)B$ pairing operations in total compared to Equation (7).

Note that batch auditing will fail if at least one incorrect auditing proof exists in all the $B$ auditing proofs. To allow most of auditing proofs to still pass the verification when there exists only a small number of incorrect auditing proofs, we can utilize binary search [5] during batch auditing.

More specifically, once the batch auditing of the $B$ auditing proofs fails, the public verifier divides the set of all the $B$ auditing proofs into two subsets, where each subset contains a number of $B/2$ auditing proofs. Then the public verifier re-checks the correctness of auditing proofs in each subset using batch auditing. If the verification result of one subset is correct, then all the auditing proofs in this subset are all correct. Otherwise, this subset is further divided into two sub-subsets, and the public verifier re-checks the correctness of auditing proofs in each sub-subset with batch auditing until all the incorrect auditing proofs are found. Clearly, when the number of incorrect auditing proofs increases, the public verifier needs more time to distinguish all the incorrect auditing proofs, and the efficiency of batch auditing will be reduced. Experimental results in Section 6 shows that, when less than 12 percent of all the $B$ auditing proofs are incorrect, batching auditing is still more efficient than verifying all the $B$ auditing proofs one by one.

## 6 Performance

In this section, we first analyze the computation and communication costs of Oruta, and then evaluate the performance of Oruta in experiments.

### 6.1 Computation Cost

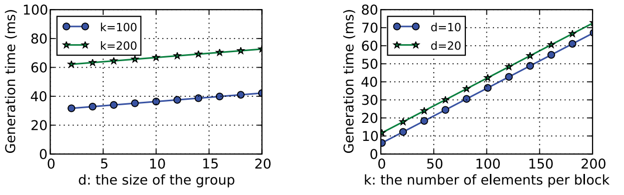
During an auditing task, the public verifier first generates some random values to construct an auditing challenge, which only introduces a small cost in computation. Then, after receiving the auditing challenge, the cloud server needs to compute an auditing proof $\{\boldsymbol{\lambda}, \boldsymbol{\mu}, \boldsymbol{\phi}, \{id_j\}_{j\in\mathcal{J}}\}$. Based on the description in Section 5, the computation cost of calculating an auditing proof is about $(k+dc)\mathrm{Exp}_{\mathbb{G}_1} + dc\mathrm{Mul}_{\mathbb{G}_1} + ck\mathrm{Mul}_{\mathbb{Z}_p} + k\mathrm{Hash}_{\mathbb{Z}_p}$, where $\mathrm{Exp}_{\mathbb{G}_1}$ denotes the cost of computing one exponentiation in $\mathbb{G}_1$, $\mathrm{Mul}_{\mathbb{G}_1}$ denotes the cost of computing one multiplication in $\mathbb{G}_1$, $\mathrm{Mul}_{\mathbb{Z}_p}$ and $\mathrm{Hash}_{\mathbb{Z}_p}$ respectively denote the cost of computing one multiplication and one hashing operation in $\mathbb{Z}_p$. To check the correctness of an auditing proof $\{\boldsymbol{\lambda}, \boldsymbol{\mu}, \boldsymbol{\phi}, \{id_j\}_{j\in\mathcal{J}}\}$, a public verifier audits it with Equation (6). The total cost of verifying this auditing proof is about $(2k+c)\mathrm{Exp}_{\mathbb{G}_1} + (2k+c)\mathrm{Mul}_{\mathbb{G}_1} + d\mathrm{Mul}_{\mathbb{G}_T} + c\mathrm{Hash}_{\mathbb{G}_1} + (d+2)\mathrm{Pair}$. We use Pair to denote the cost of computing one pairing operation on $e: \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$.

### 6.2 Communication Cost

The communication cost of Oruta is mainly introduced by two aspects: the auditing challenge and auditing proof. For each auditing challenge $\{j, y_j\}_{j\in\mathcal{J}}$, the communication cost is $c(|q| + |n|)$ bits, where $|q|$ is the length of an element of $\mathbb{Z}_q$ and $|n|$ is the length of an index. Each auditing proof $\{\boldsymbol{\lambda}, \boldsymbol{\mu}, \boldsymbol{\phi}, \{id_j\}_{j\in\mathcal{J}}\}$ contains $(k+d)$ elements of $\mathbb{G}_1$, $k$ elements of $\mathbb{Z}_p$ and $c$ elements of $\mathbb{Z}_q$, therefore the communication cost of one auditing proof is $(2k+d)|p| + c|q|$ bits.

### 6.3 Experimental Results

We now evaluate the efficiency of Oruta in experiments. In our experiments, we utilize the GNU Multiple Precision Arithmetic (GMP) library and Pairing Based Cryptography (PBC) library. All the following experiments are based on C and tested on a 2.26 GHz Linux system over $1{,}000$ times. Because Oruta needs more exponentiations than pairing operations during the process of auditing, the elliptic curve we choose in our experiments is an MNT curve with a base field size of 159 bits, which has a better performance than other curves on computing exponentiations. We choose $|p| = 160$ bits and $|q| = 80$ bits. We assume the total number of blocks in shared data is $n = 1{,}000{,}000$ and $|n| = 20$ bits. The size of shared data is 2 GB. To keep the detection probability greater than 99 percent, we set the number of selected



(a) Impact of $d$ on signature generation time (ms).

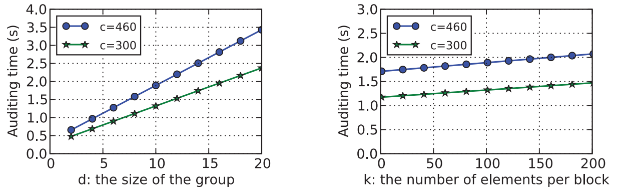
(b) Impact of $k$ on signature generation time (ms).

Fig. 10. Performance of signature generation.

blocks in an auditing task as $c = 460$ [9]. If only 300 blocks are selected, the detection probability is greater than 95 percent. We also assume the size of the group $d \in [2, 20]$ in the following experiments. Certainly, if a larger group size is used, the total computation cost will increase due to the increasing number of exponentiations and pairing operations.

*Performance of Signature Generation.* According to Section 5, the generation time of a ring signature on a block is determined by the number of users in the group and the number of elements in each block. As illustrated in Figs. 10a and 10b, when $k$ is fixed, the generation time of a ring signature is linearly increasing with the size of the group; when $d$ is fixed, the generation time of a ring signature is linearly increasing with the number of elements in each block. Specifically, when $d = 10$ and $k = 100$, a user in the group requires about 37 milliseconds to compute a ring signature on a block in shared data.

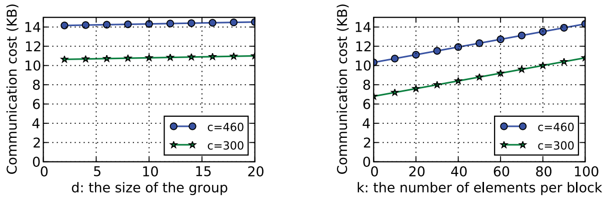
*Performance of Auditing.* Based on our proceeding analyses, the auditing performance of Oruta under different detection probabilities is illustrated in Figs. 11a and 12b, and Table 2. As shown in Fig. 11a, the auditing time is linearly increasing with the size of the group. When $c = 300$, if there are two users sharing data in the cloud, the auditing time is only about $0.5$ seconds; when the number of group member increases to 20, it takes about $2.5$ seconds to finish the same auditing task. The communication cost of an auditing task under different parameters is presented in Figs. 12a and 12b. Compared to the size of entire shared data, the communication cost that a public verifier consumes in an auditing task is very small. It is clear in Table 2 that when maintaining a higher detection probability, a public verifier needs to consume more computation and communication overhead to finish the auditing task. Specifically, when $c = 300$, it takes a public verifier $1.32$ seconds to audit the correctness of shared data, where the size of shared data is 2 GB; when $c = 460$, a public verifier needs $1.94$ seconds to verify the integrity of the same shared data.



(a) Impact of $d$ on auditing time (second), where $k = 100$.

(b) Impact of $k$ on auditing time (second), where $d = 10$.

Fig. 11. Performance of auditing time.



(a) Impact of $d$ on communication cost (KB), where $k = 100$.

(b) Impact of $k$ on communication cost (KB), where $d = 10$.

Fig. 12. Performance of communication cost.

As we discussed in the previous section, the privacy performance of our mechanism depends on the number of members in the group. Given a block in shared data, the probability that a public verifier fails to reveal the identity of the signer is $1 - 1/d$, where $d \geq 2$. Clearly, when the number of group members is larger, our mechanism has a better performance in terms of privacy. As we can see from Fig. 13a, this privacy performance increases with an increase of the size of the group.

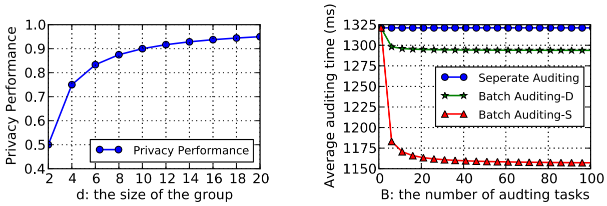
*Performance of Batch Auditing.* As we discussed in Section 5, when there are multiple auditing proofs, the public verifier can improve the efficiency of verification by performing batch auditing. In the following experiments, we choose $c = 300$, $k = 100$ and $d = 10$. Compared to verifying a number of $B$ auditing proofs one by one, if these $B$ auditing proofs are for different groups, batching auditing can save $2.1$ percent of the auditing time per auditing proof on average (as shown in Fig. 14a). If these $B$ auditing tasks are for the same group, batching auditing can save $12.6$ percent of the average auditing time per auditing proof (as shown in Fig. 14b).

Now we evaluate the performance of batch auditing when incorrect auditing proofs exist among the $B$ auditing proofs. As we mentioned in Section 5, we can use binary search in batch auditing, so that we can distinguish the incorrect ones from the $B$ auditing proofs. However, the increasing number of incorrect auditing proofs will reduce the efficiency of batch auditing. It is important for us to find out the maximal number of incorrect auditing proofs exist in the $B$ auditing proofs, where the batch auditing is still more efficient than separate auditing.

In this experiment, we assume the total number of auditing proofs in the batch auditing is $B = 128$ (because we leverage binary search, it is better to set $B$ as a power of 2), the number of elements in each block is $k = 100$ and the number of users in the group is $d = 10$. Let $A$ denote the number of incorrect auditing proofs. In addition, we also assume that it always requires the *worst-case* algorithm to detect the incorrect auditing proofs in the experiment. According to Equation (7) and (8), the extra computation cost in binary search is mainly

TABLE 2
Performance of Auditing

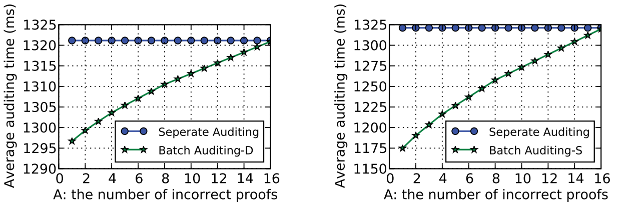
| System Parameters | $k = 100$, $d = 10$, | |
|---|---|---|
| Storage Usage | 2GB + 200MB (data + signatures) | |
| Selected Blocks $c$ | 460 | 300 |
| Communication Cost | 14.55KB | 10.95KB |
| Auditing Time | 1.94s | 1.32s |



(a) Impact of $d$ on privacy performance.

(b) Impact of $B$ on the efficiency of batch auditing, where $k = 100$ and $d = 10$.

Fig. 13. Performance of privacy and batch auditing.



(a) Impact of $A$ on the efficiency of batch auditing, where $B = 128$.

(b) Impact of $A$ on the efficiency of batch auditing, where $B = 128$.

Fig. 14. Efficiency of batch auditing with incorrect proofs.

introduced by extra pairing operations. As shown in Fig. 14a, if all the 128 auditing proofs are for different groups, when the number of incorrect auditing proofs is less than 16 (12 percent of all the auditing proofs), batching auditing is still more efficient than separate auditing. Similarly, in Fig. 14b, if all the auditing proofs are for the same group, when the number of incorrect auditing proofs is more than 16, batching auditing is less efficient than verifying these auditing proofs separately.

## 7 Related Work

Provable data possession (PDP), proposed by Ateniese et al. [9], allows a verifier to check the correctness of a client's data stored at an untrusted server. By utilizing RSA-based homomorphic authenticators and sampling strategies, the verifier is able to publicly audit the integrity of data without retrieving the entire data, which is referred to as public auditing. Unfortunately, their mechanism is only suitable for auditing the integrity of personal data. Juels and Kaliski [32] defined another similar model called Proofs of Retrievability (POR), which is also able to check the correctness of data on an untrusted server. The original file is added with a set of randomly-valued check blocks called *sentinels*. The verifier challenges the untrusted server by specifying the positions of a collection of sentinels and asking the untrusted server to return the associated sentinel values. Shacham and Waters [10] designed two improved schemes. The first scheme is built from BLS signatures [27], and the second one is based on pseudo-random functions.

To support dynamic data, Ateniese et al. [33] presented an efficient PDP mechanism based on symmetric keys. This mechanism can support update and delete operations on data, however, insert operations are not available in this mechanism. Because it exploits symmetric keys to verify the integrity of data, it is not public verifiable and only provides a user with a limited number of verification requests. Wang et al. [12] utilized Merkle Hash Tree and BLS signatures [27] to support dynamic data in a public auditing mechanism. Erway et al. [11] introduced dynamic provable data possession (DPDP) by using authenticated dictionaries, which are based on rank information. Zhu et al. [15] exploited the fragment structure to reduce the storage of signatures in their public auditing mechanism. In addition, they also used index hash tables to provide dynamic operations on data. The public mechanism proposed by Wang et al. [5] and its journal version [18] are able to preserve users' confidential data from a public verifier by using random maskings. In addition, to operate multiple auditing tasks from different users efficiently, they extended their mechanism to enable batch auditing by leveraging aggregate signatures [21].

Wang et al. [13] leveraged homomorphic tokens to ensure the correctness of erasure codes-based data distributed on multiple servers. This mechanism is able not only to support dynamic data, but also to identify misbehaved servers. To minimize communication overhead in the phase of data repair, Chen et al. [14] also introduced a mechanism for auditing the correctness of data under the multi-server scenario, where these data are encoded by network coding instead of using erasure codes. More recently, Cao et al. [16] constructed an LT codes-based secure and reliable cloud storage mechanism. Compare to previous work [13], [14], this mechanism can avoid high decoding computation cost for data users and save computation resource for online data owners during data repair.

## 8 Conclusion and Future Work

In this paper, we propose Oruta, a privacy-preserving public auditing mechanism for shared data in the cloud. We utilize ring signatures to construct homomorphic authenticators, so that a public verifier is able to audit shared data integrity without retrieving the entire data, yet it cannot distinguish who is the signer on each block. To improve the efficiency of verifying multiple auditing tasks, we further extend our mechanism to support batch auditing.

There are two interesting problems we will continue to study for our future work. One of them is traceability, which means the ability for the group manager (i.e., the original user) to reveal the identity of the signer based on verification metadata in some special situations. Since Oruta is based on ring signatures, where the identity of the signer is unconditionally protected [21], the current design of ours does not support traceability. To the best of our knowledge, designing an efficient public auditing mechanism with the capabilities of preserving identity privacy and supporting traceability is still open. Another problem for our future work is how to prove data freshness (prove the cloud possesses the latest version of shared data) while still preserving identity privacy.

## Acknowledgments

This work was supported by NSFC 61272457, National Project 2012ZX03002003-002, 863 Project 2012AA013102, 111 Project B08038, IRT1078, FRF K50511010001 and NSFC 61170251. Most part of this work was done at University of

Toronto. A short version [1] of this paper is in Proceedings of the Fifth IEEE International Conference on Cloud Computing (IEEE Cloud 2012).

## REFERENCES

[1] B. Wang, B. Li, and H. Li, "Oruta: Privacy-Preserving Public Auditing for Shared Data in the Cloud," *Proc. IEEE Fifth Int'l Conf. Cloud Computing*, pp. 295-302, 2012.

[2] 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," *Comm. ACM*, vol. 53, no. 4, pp. 50-58, Apr. 2010.

[3] K. Ren, C. Wang, and Q. Wang, "Security Challenges for the Public Cloud," *IEEE Internet Computing*, vol. 16, no. 1, pp. 69-73, 2012.

[4] D. Song, E. Shi, I. Fischer, and U. Shankar, "Cloud Data Protection for the Masses," *Computer*, vol. 45, no. 1, pp. 39-45, 2012.

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

[6] B. Wang, M. Li, S.S. Chow, and H. Li, "Computing Encrypted Cloud Data Efficiently under Multiple Keys," *Proc. IEEE Conf. Comm. and Network Security (CNS '13)*, pp. 90-99, 2013.

[7] R. Rivest, A. Shamir, and L. Adleman, "A Method for Obtaining Digital Signatures and Public Key Cryptosystems," *Comm. ACM*, vol. 21, no. 2, pp. 120-126, 1978.

[8] The MD5 Message-Digest Algorithm (RFC1321). https://tools.ietf.org/html/rfc1321, 2014.

[9] G. Ateniese, R. Burns, R. Curtmola, J. Herring, L. Kissner, Z. Peterson, and D. Song, "Provable Data Possession at Untrusted Stores," *Proc. 14th ACM Conf. Computer and Comm. Security (CCS '07)*, pp. 598-610, 2007.

[10] H. Shacham and B. Waters, "Compact Proofs of Retrievability," *Proc. 14th Int'l Conf. Theory and Application of Cryptology and Information Security: Advances in Cryptology (ASIACRYPT '08)*, pp. 90-107, 2008.

[11] C. Erway, A. Kupcu, C. Papamanthou, and R. Tamassia, "Dynamic Provable Data Possession," *Proc. 16th ACM Conf. Computer and Comm. Security (CCS'09)*, pp. 213-222, 2009.

[12] Q. Wang, C. Wang, J. Li, K. Ren, and W. Lou, "Enabling Public Verifiability and Data Dynamic for Storage Security in Cloud Computing," *Proc. 14th European Conf. Research in Computer Security (ESORICS'09)*, pp. 355-370, 2009.

[13] C. Wang, Q. Wang, K. Ren, and W. Lou, "Ensuring Data Storage Security in Cloud Computing," *Proc. 17th Int'l Workshop Quality of Service (IWQoS'09)*, pp. 1-9, 2009.

[14] B. Chen, R. Curtmola, G. Ateniese, and R. Burns, "Remote Data Checking for Network Coding-Based Distributed Storage Systems," *Proc. ACM Workshop Cloud Computing Security Workshop (CCSW'10)*, pp. 31-42, 2010.

[15] Y. Zhu, H. Wang, Z. Hu, G.-J. Ahn, H. Hu, and S.S Yau, "Dynamic Audit Services for Integrity Verification of Outsourced Storages in Clouds," *Proc. ACM Symp. Applied Computing (SAC'11)*, pp. 1550-1557, 2011.

[16] N. Cao, S. Yu, Z. Yang, W. Lou, and Y.T. Hou, "LT Codes-Based Secure and Reliable Cloud Storage Service," *Proc. IEEE INFOCOM*, 2012.

[17] B. Wang, B. Li, and H. Li, "Certificateless Public Auditing for Data Integrity in the Cloud," *Proc. IEEE Conf. Comm. and Network Security (CNS'13)*, pp. 276-284, 2013.

[18] C. Wang, S.S. Chow, Q. Wang, K. Ren, and W. Lou, "Privacy-Preserving Public Auditing for Secure Cloud Storage," *IEEE Trans. Computers*, vol. 62, no. 2, pp. 362-375, Feb. 2013.

[19] B. Wang, B. Li, and H. Li, "Public Auditing for Shared Data with Efficient User Revocation in the Cloud," *Proc. IEEE INFOCOM*, pp. 2904-2912, 2013.

[20] B. Wang, B. Li, and H. Li, "Panda: Public Auditing for Shared Data with Efficient User Revocation in the Cloud," *IEEE Trans. Services Computing*, 20 Dec. 2013, DOI: 10.1109/TSC.2013.2295611.

[21] D. Boneh, C. Gentry, B. Lynn, and H. Shacham, "Aggregate and Verifiably Encrypted Signatures from Bilinear Maps," *Proc. 22nd Int'l Conf. Theory and Applications of Cryptographic Techniques: Advances in Cryptology (EUROCRYPT'03)*, pp. 416-432, 2003.

[22] B. Wang, H. Li, and M. Li, "Privacy-Preserving Public Auditing for Shared Cloud Data Supporting Group Dynamics," *Proc. IEEE Int'l Conf. Comm. (ICC'13)*, pp. 539-543, 2013.

[23] B. Wang, S.S. Chow, M. Li, and H. Li, "Storing Shared Data on the Cloud via Security-Mediator," *Proc. IEEE 33rd Int'l Conf. Distributed Computing Systems (ICDCS'13)*, pp. 124-133, 2013.

[24] D. Boneh, X. Boyen, and H. Shacham, "Short Group Signatures," *Proc. 24th Ann. Int'l Cryptology Conf. (CRYPTO'04)*, pp. 41-55, 2004.

[25] B. Wang, B. Li, and H. Li, "Knox: Privacy-Preserving Auditing for Shared Data with Large Groups in the Cloud," *Proc. 10th Int'l Conf. Applied Cryptography and Network Security (ACNS'12)*, pp. 507-525, June 2012.

[26] E. Brickell, J. Camenisch, and L. Chen, "Direct Anonymous Attestation," *Proc. 11th ACM Conf. Computer and Comm. Security (CCS'04)*, pp. 132-145, 2004.

[27] D. Boneh, B. Lynn, and H. Shacham, "Short Signatures from the Weil Pairing," *Proc. Seventh Int'l Conf. Theory and Application of Cryptology and Information Security: Advances in Cryptology (ASIACRYPT'01)*, pp. 514-532, 2001.

[28] R.L. Rivest, A. Shamir, and Y. Tauman, "How to Leak a Secret," *Proc. Seventh Int'l Conf. Theory and Application of Cryptology and Information Security: Advances in Cryptology (ASIACRYPT'01)*, pp. 552-565, 2001.

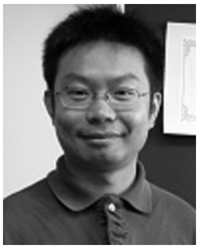
[29] D. Cash, A. Kupcu, and D. Wichs, "Dynamic Proofs of Retrievability via Oblivious RAM," *Proc. 32nd Ann. Int'l Conf. Theory and Applications of Cryptographic Techniques: Advances in Cryptology (EUROCRYPT)*, pp. 279-295, 2013.

[30] S. Yu, C. Wang, K. Ren, and W. Lou, "Achieving Secure, Scalable, and Fine-Grained Data Access Control in Cloud Computing," *Proc. IEEE INFOCOM*, pp. 534-542, 2010.

[31] X. Liu, Y. Zhang, B. Wang, and J. Yan, "Mona: Secure Multi-Owner Data Sharing for Dynamic Groups in the Cloud," *IEEE Trans. Parallel and Distributed Systems*, vol. 24, no. 6, pp. 1182-1191, June 2013.

[32] A. Juels and B.S. Kaliski, "PORs: Proofs of Retrievability for Large Files," *Proc. 14th ACM Conf. Computer and Comm. Security (CCS'07)*, pp. 584-597, 2007.

[33] G. Ateniese, R.D. Pietro, L.V. Mancini, and G. Tsudik, "Scalable and Efficient Provable Data Possession," *Proc. Fourth Int'l Conf. Security and Privacy in Comm. Networks (SecureComm'08)*, 2008.



**Boyang Wang** received the BS degree in information security from Xidian University in 2007. He is currently working toward the PhD degree from the School of Telecommunications Engineering, Xidian University, Xi'an, China. His current research interests include focus on security and privacy issues in cloud computing, big data, and applied cryptography. He is a student member of the IEEE.

**Baochun Li** is a professor at the Department of Electrical and Computer Engineering at the University of Toronto, and holds the Bell University Laboratories Endowed chair in computer engineering. His research interests include large-scale multimedia systems, cloud computing, peer-to-peer networks, applications of network coding, and wireless networks. He is a member of the ACM and a senior member of the IEEE.

**Hui Li** received the BSc degree from Fudan University in 1990, the MSc and the PhD degrees from Xidian University in 1993 and 1998, respectively. He is a professor at the School of Telecommunications Engineering, Xidian University, Xi'an, China. In 2009, he was with the Department of ECE, University of Waterloo as a visiting scholar. His research interests include cryptography, security of cloud computing, wireless network security, and information theory. He is a member of the IEEE.