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

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

Abstract—With cloud storage services, it is commonplace for data to be not only stored in the cloud, but also shared across multiple users. However, public auditing for such shared data — while preserving identity privacy — remains to be an open challenge. In this paper, we propose the first privacy-preserving mechanism that allows public auditing on shared data stored in the cloud. In particular, we exploit ring signatures to compute the verification information needed to audit the integrity of shared data. With our mechanism, the identity of the signer on each block in shared data is kept private from a third party auditor (TPA), who is still able to publicly verify the integrity of shared data without retrieving the entire file. Our experimental results demonstrate the effectiveness and efficiency of our proposed mechanism when auditing shared data.

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

1 Introduction

Cloud service providers manage an enterprise-class infrastructure that offers a scalable, secure and reliable environment for users, at a much lower marginal cost due to the sharing nature of resources. It is routine for users to use cloud storage services to share data with others in a team, as data sharing becomes a standard feature in most cloud storage offerings, including Dropbox and Google Docs.

The integrity of data in cloud storage, however, is subject to skepticism and scrutiny, as data stored in an untrusted cloud can easily be lost or corrupted, due to hardware failures and human errors [1]. To protect the integrity of cloud data, it is best to perform public auditing by introducing a third party auditor (TPA), who offers its auditing service with more powerful computation and communication abilities than regular users.

The first provable data possession (PDP) mechanism [2] to perform public auditing is designed to check the correctness of data stored in an untrusted server, without retrieving the entire data. Moving a step forward, Wang *et al.* [3] (referred to as WWRL in this paper) is designed to construct a public auditing mechanism for cloud data, so that during public auditing, the content of private data belonging to a personal user is not disclosed to the third party auditor.

We believe that sharing data among multiple users is perhaps one of the most engaging features that motivates cloud storage. A unique problem introduced during the process of public auditing for shared data in the cloud is how to preserve **identity privacy** from the TPA, because the identities of signers on shared data may indicate that a particular user in the group or a special block in shared data is a higher valuable target than others.

For example, Alice and Bob work together as a group and share a file in the cloud. The shared file is divided into a number of small blocks, which are independently signed by users. Once a block in this shared file is modified by a user, this user needs to sign the new block using her public/private key pair. The TPA needs to know the identity of the signer on each block in this shared file, so that it is able to audit the integrity of the whole file based on requests from Alice or Bob.

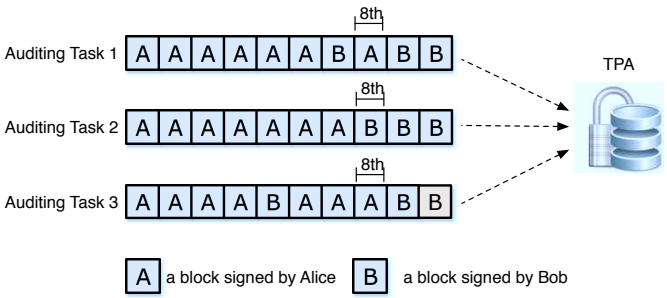

Fig. 1. Alice and Bob share a file in the cloud. The TPA audits the integrity of shared data with existing mechanisms.

As shown in Fig. 1, after performing several auditing tasks, some private and sensitive information may reveal to the TPA. On one hand, most of the blocks in shared file are signed by Alice, which may indicate that Alice is a important role in this group, such as a group leader. On the other hand, the 8-th block is frequently modified by different users. It means this block may contain high-value data, such as a final bid in an auction, that Alice

- *Boyang Wang and Hui Li are with the State Key Laboratory of Integrated Services Networks, Xidian University, Xi'an, 710071, China. Boyang Wang is also a visiting Ph.D student at Department of Electrical and Computer Engineering, University of Toronto.*
  E-mail: {bywang,lihui}@mail.xidian.edu.cn
- *Baochun Li is with the Department of Electrical and Computer Engineering, University of Toronto, Toronto, ON, M5S 3G4, Canada.*
  E-mail: bli@eecg.toronto.edu

and Bob need to discuss and change it several times.

As described in the example above, the identities of signers on shared data may indicate which user in the group or block in shared data is a higher valuable target than others. Such information is confidential to the group and should not be revealed to any third party. However, no existing mechanism in the literature is able to perform public auditing on shared data in the cloud while still preserving identity privacy.

In this paper, we propose Oruta[1], a new privacy-preserving public auditing mechanism for shared data in an untrusted cloud. In Oruta, we utilize ring signatures [4], [5] to construct homomorphic authenticators [2], [6], so that the third party auditor is able to verify the integrity of shared data for a group of users without retrieving the entire data — while the identity of the signer on each block in shared data is kept private from the TPA. In addition, we further extend our mechanism to support batch auditing, which can audit multiple shared data simultaneously in a single auditing task. Meanwhile, Oruta continues to use random masking [3] to support data privacy during public auditing, and leverage index hash tables [7] to support fully dynamic operations on shared data. A dynamic operation indicates an insert, delete or update operation on a single block in shared data. A high-level comparison between Oruta and existing mechanisms in the literature is shown in Table 1. To our best knowledge, this paper represents the first attempt towards designing an effective privacy-preserving public auditing mechanism for shared data in the cloud.

TABLE 1
Comparison with Existing Mechanisms

	PDP [2]	WWRL [3]	Oruta
Public auditing	Yes	Yes	Yes
Data privacy	No	Yes	Yes
Identity privacy	No	No	Yes

The remainder of this paper is organized as follows. In Section 2, we present the system model and threat model. 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 evaluates 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, our work in this paper involves three parties: the cloud server, the third party auditor (TPA) and users. There are two types of users in a group: the original user and a number of group users. The original user and group users are both members of the group. Group members are allowed to access and modify shared data created by the original user based on access control polices [8]. Shared data and its verification information (i.e. signatures) are both stored in the cloud server. The third party auditor is able to verify the integrity of shared data in the cloud server on behalf of group members.

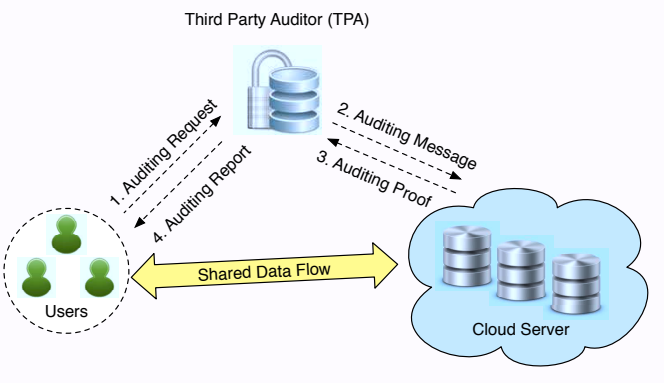

Fig. 2. Our system model includes the cloud server, the third party auditor and users.

In this paper, we only consider how to audit the integrity of shared data in the cloud with **static** groups. It means the group is pre-defined before shared data is created in the cloud and the membership of users in the group is not changed during data sharing. The original user is responsible for deciding who is able to share her data before outsourcing data to the cloud. Another interesting problem is how to audit the integrity of shared data in the cloud with **dynamic** groups — a new user can be added into the group and an existing group member can be revoked during data sharing — while still preserving identity privacy. We will leave this problem to our future work.

When a user (either the original user or a group user) wishes to check the integrity of shared data, she first sends an auditing request to the TPA. After receiving the auditing request, the TPA generates an auditing message to the cloud server, and retrieves an auditing proof of shared data from the cloud server. Then the TPA verifies the correctness of the auditing proof. Finally, the TPA sends an auditing report to the user based on the result of the verification.

2.2 Threat Model

2.2.1 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 and prevent users from using data correctly. 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, in order to avoid jeopardizing its reputation, the cloud server provider may be reluctant to inform users about such corruption of data.

1. Oruta: One Ring to Rule Them All.

2.2.2 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 **semi-trusted** TPA, who is only responsible for auditing the integrity of shared data, may try to reveal the identity of the signer on each block in shared data based on verification information. Once the TPA 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).

2.3 Design Objectives

To enable the TPA efficiently and securely verify shared data for a group of users, Oruta should be designed to achieve following properties: (1) **Public Auditing**: The third party auditor is able to publicly verify the integrity of shared data for a group of users without retrieving the entire data. (2) **Correctness**: The third party auditor is able to correctly detect whether there is any corrupted block in shared data. (3) **Unforgeability**: Only a user in the group can generate valid verification information on shared data. (4) **Identity Privacy**: During auditing, the TPA cannot distinguish the identity of the signer on each block in shared data.

3 Preliminaries

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

3.1 Bilinear Maps

We first introduce a few concepts and properties related to bilinear maps. We follow notations from [5], [9]:

1) G_1, G_2 and G_T are three multiplicative cyclic groups of prime order p;
2) g_1 is a generator of G_1, and g_2 is a generator of G_2;
3) ψ is a computable isomorphism from G_2 to G_1, with $\psi(g_2) = g_1$;
4) e is a bilinear map e: $G_1 \times G_2 \rightarrow G_T$ with the following properties: **Computability**: there exists an efficiently computable algorithm for computing the map e. **Bilinearity**: for all $u \in G_1$, $v \in G_2$ and $a, b \in Z_p$, $e(u^a, v^b) = e(u, v)^{ab}$. **Non-degeneracy**: $e(g_1, g_2) \neq 1$.

These properties further imply two additional properties: (1) for any u_1, $u_2 \in G_1$ and $v \in G_2$, $e(u_1 \cdot u_2, v) = e(u_1, v) \cdot e(u_2, v)$; (2) for any u, $v \in G_2$, $e(\psi(u), v) = e(\psi(v), u)$.

3.2 Complexity Assumptions

Definition 1: **Discrete Logarithm Problem.** For $a \in Z_p$, given $g, h = g^a \in G_1$, output a.

The Discrete Logarithm assumption holds in G_1 if no t-time algorithm has advantage at least ϵ in solving the Discrete Logarithm problem in G_1, which means it is computational infeasible to solve the Discrete Logarithm problem in G_1.

Definition 2: **Computational Co-Diffie-Hellman Problem.** For $a \in Z_p$, given $g_2, g_2^a \in G_2$ and $h \in G_1$, compute $h^a \in G_1$.

The co-CDH assumption holds in G_1 and G_2 if no t-time algorithm has advantage at least ϵ in solving the co-CDH problem in G_1 and G_2. When $G_1 = G_2$ and $g_1 = g_2$, the co-CDH problem can be reduced to the standard CDH problem in G_1. The co-CDH assumption is a stronger assumption than the Discrete Logarithm assumption.

Definition 3: **Computational Diffie-Hellman Problem.** For $a, b \in Z_p$, given $g_1, g_1^a, g_1^b \in G_1$, compute $g_1^{ab} \in G_1$.

The CDH assumption holds in G_1 if no t-time algorithm has advantage at least ϵ in solving the CDH problem in G_1.

3.3 Ring Signatures

The concept of ring signatures is first proposed by Rivest *et al.* [4] 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. This property can be used to preserve the identity of the signer from a verifier.

The ring signature scheme introduced by Boneh *et al.* [5] (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 data auditing mechanisms [2], [3], [6]. Besides unforgeability (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, σ_1 denote a signature on block $m_1 \in Z_p$, σ_2 denote a signature on block $m_2 \in Z_p$.

- **Blockless verification:** Given σ_1 and σ_2, two random values α_1, $\alpha_2 \in Z_p$ and a block $m' = \alpha_1 m_1 + \alpha_2 m_2 \in Z_p$, a verifier is able to check the correctness of block m' without knowing block m_1 and m_2.
- **Non-malleability** Given σ_1 and σ_2, two random values $\alpha_1, \alpha_2 \in Z_p$ and a block $m' = \alpha_1 m_1 + \alpha_2 m_2 \in Z_p$, a user, who does not have private key **sk**, is not able to generate a valid signature σ' on block m' by linearly combining signature σ_1 and σ_2.

Blockless verification allows a verifier to audit the correctness of data stored in the cloud server with a single block, which is a linear combination of all the blocks in data. If the combined block is correct, the verifier

believes that the blocks in data are all 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 attacker cannot generate valid signatures on invalid blocks by linearly combining existing signatures.

Other cryptographic techniques related to homomorphic authenticable signatures includes aggregate signatures [5], homomorphic signatures [10] and batch-verification signatures [11]. If a signature scheme is blockless verifiable and malleable, it is a homomorphic signature scheme. In the construction of data auditing mechanisms, we should use homomorphic authenticable signatures, not homomorphic signatures.

4 Homomorphic Authenticable Ring Signatures

4.1 Overview

In this section, we introduce a new ring signature scheme, which is suitable for public auditing. Then, 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.

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 the TPA. However, traditional ring signatures [4], [5] cannot be directly used into public auditing mechanisms, because these ring signature schemes do not support blockless verification. Without blockless verification, the TPA has to download the whole data file to verify the correctness of shared data, which consumes excessive bandwidth and takes long verification times.

Therefore, we first construct a new homomorphic authenticable ring signature (HARS) scheme, which is extended from a classic ring signature scheme [5], denoted as BGLS. The ring signatures generated by HARS is able not only to preserve identity privacy but also to support blockless verification.

4.2 Construction of HARS

HARS contains three algorithms: **KeyGen**, **RingSign** and **RingVerify**. In **KeyGen**, each user in the group generates her public key and private key. In **RingSign**, a user in the group is able to sign a block with her private key and all the group members' public keys. A verifier is allowed to check whether a given block is signed by a group member in **RingVerify**.

Scheme Details. Let G_1, G_2 and G_T be multiplicative cyclic groups of order p, g_1 and g_2 be generators of G_1 and G_2 respectively. Let $e : G_1 \times G_2 \to G_T$ be a bilinear map, and $\psi : G_2 \to G_1$ be a computable isomorphism with $\psi(g_2) = g_1$. There is a public map-to-point hash function H_1: $\{0,1\}^* \to G_1$. The global parameters are $(e, \psi, p, G_1, G_2, G_T, g_1, g_2, H_1)$. The total number of users in the group is d. Let U denote the group that includes all the d users.

KeyGen. For a user u_i in the group U, she randomly picks $x_i \in Z_p$ and computes $w_i = g_2^{x_i} \in G_2$. Then, user u_i's public key is $\mathbf{pk}_i = w_i$ and 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 $m \in Z_p$, the identifier of this block id and the private key $\mathbf{sk}_s$ for some s, user u_s randomly chooses $a_i \in Z_p$ for all $i \neq s$, where $i \in [1, d]$, and let $\sigma_i = g_1^{a_i}$. Then, she computes

$$\beta = H_1(id) g_1^m \in G_1, \tag{1}$$

and sets

$$\sigma_s = \left(\frac{\beta}{\psi(\prod_{i \neq s} w_i^{a_i})} \right)^{1/x_s} \in G_1. \tag{2}$$

And the ring signature of block m is $\boldsymbol{\sigma} = (\sigma_1, ..., \sigma_d) \in G_1^d$.

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 $\beta = H_1(id) g_1^m \in G_1$, and then checks

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

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.

4.3 Security Analysis of HARS

Now, we discuss some important properties of HARS, including correctness, unforgeability, blockless verification, non-malleability and identity privacy.

***Theorem** 1: Given any block and its ring signature, a verifier is able to correctly check the integrity of this block under HARS.*

Proof: To prove the correctness of HARS is equivalent of proving Equation (3) is correct. Based on properties of bilinear maps, the correctness of this equation can be proved as follows:

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

□

Now we prove that HARS is able to resistance to forgery. We follow the security model and the game defined in BGLS [5]. In the game, an adversary is given all the d users' public key $(\mathbf{pk}_1, ..., \mathbf{pk}_d) = (w_1, ..., w_d)$, and is given access to the hash oracle and the ring-signing oracle. The goal of the adversary is to output a valid ring signature on a pair of block/identifier (id, m), where this pair of block/identifier (id, m) has never been presented to the ring-signing oracle. If the adversary achieves this goal, then it wins the game.

***Theorem** 2: Suppose $\mathcal{A}$ is a (t', ϵ')-algorithm that can generate a forgery of a ring signature on a group of users of size d. Then there exists a (t, ϵ)-algorithm that can solve the co-CDH problem with $t \leq 2t' + 2c_{G_1}(q_H + dq_s + q_s + d) + 2c_{G_2}d$ and $\epsilon \geq (\epsilon'/(e + eq_s))^2$, where $\mathcal{A}$ issues at most q_H hash queries and at most q_s ring-signing queries, $e = \lim_{q_s \to \infty}(1 + 1/q_s)^{q_s}$, exponentiation and inversion on G_1 take time c_{G_1}, and exponentiation and inversion on G_2 take time c_{G_2}.*

Proof: The co-CDH problem can be solved by solving two random instances of the following problem: Given g_1^{ab}, g_2^a (and g_1,g_2), compute g_1^b. We shall construct an algorithm $\mathcal{B}$ that solves this problem. This problem is easy if $a = 0$. In what follows, we assume $a \neq 0$.

Initially, $\mathcal{B}$ randomly picks $x_2, ..., x_n$ from Z_p and sets $x_1 = 1$. Then, it sets $\mathbf{pk}_i = w_i = (g_2^a)^{x_i}$. Algorithm $\mathcal{A}$ is given the public keys $(\mathbf{pk}_1, ..., \mathbf{pk}_d) = (w_1, ..., w_d)$. 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.

On a hash query, $\mathcal{B}$ flips a coin that shows 0 with probability p_c and 1 otherwise, where p_c will be determined later. Then $\mathcal{B}$ randomly picks $r \in Z_p$, if the coin shows 0, $\mathcal{B}$ returns $(g_1^{ab})^r$, otherwise it returns $\psi(g_2^a)^r$.

Suppose $\mathcal{A}$ issues a ring sign query on a block m and its identifier id. By the assumption, a hash query has been issued 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. In this case, $\mathcal{B}$ chooses random $a_2, ..., a_d \in Z_p$, computes $a_1 = r - (a_2x_2 + ... + a_dx_d)$, and returns the signature $\boldsymbol{\sigma} = (g_1^{a_1}, ..., g_1^{a_d})$.

Eventually $\mathcal{A}$ outputs a forgery $\boldsymbol{\sigma} = (\sigma_1, ..., \sigma_d)$ 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 chosen by $\mathcal{B}$, and $\mathcal{B}$ can output g_1^b by computing $(\prod_{i=1}^d \sigma_i^{x_i})^{1/r}$.

Algorithm $\mathcal{A}$ cannot distinguish between $\mathcal{B}$'s simulation and real life. If $\mathcal{A}$ successfully forges a ring signature, then $\mathcal{B}$ can output g_1^b. The probability that $\mathcal{B}$ will not fail is $p_c^{q_s}(1 - p)$, which is maximized when $p_c = q_s/(q_s + 1)$, then the bound of this probability is $1/(e \cdot (1 + q_s))$, where $e = \lim_{q_s \to \infty}(1 + 1/q_s)^{q_s}$. Algorithm $\mathcal{B}$ requires d exponentiations on G_2 in setup, one exponentiations on G_1 for each of $\mathcal{A}$'s hash queries, $d + 1$ exponentiations on G_1 for each of $\mathcal{A}$'s signature queries, and d exponentiations on G_1 in the output phase, so the running time of $\mathcal{B}$ is the running time of $\mathcal{A}$ plus $c_{G_1}(q_H + dq_s + q_s + d) + c_{G_2}d$.

Because the co-CDH problem can be solved by solving two random instances of algorithm $\mathcal{B}$. Therefore, if $\mathcal{A}$ is a (t', ϵ')-algorithm that can generate a forgery of ring signature on a group of users of size d, then there exists a (t, ϵ)-algorithm that can solve the co-CDH problem with $t \leq 2t' + 2c_{G_1}(q_H + dq_s + q_s + d) + 2c_{G_2}d$ and $\epsilon \geq (\epsilon'/(e + eq_s))^2$. □

***Theorem** 3: For an adversary, it is computational infeasible to forge a ring signature under HARS.*

Proof: As we already proved in Theorem 2, if an adversary can forge a ring signature, then we can find a (t, ϵ)-algorithm to solve the co-CDH problem in G_1 and G_2, which contradicts to the fact that the co-CDH problem in G_1 and G_2 is hard. Therefore, for an adversary, it is computational infeasible to forge a ring signature under HARS. □

Then, based on Theorem 1 and 3, we show that HARS is a homomorphic authenticable ring signature scheme.

***Theorem** 4: 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 verification, which we defined in Section 3. Then we show HARS is also non-malleable.

Given all the d users' public keys $(\mathbf{pk}_1, ..., \mathbf{pk}_d) = (w_1, ..., w_d)$, two identifiers id_1 and id_2, two ring signatures $\boldsymbol{\sigma}_1 = (\sigma_{1,1}, ..., \sigma_{1,d})$ and $\boldsymbol{\sigma}_2 = (\sigma_{2,1}, ..., \sigma_{2,d})$, and two random values y_1, $y_2 \in Z_p$, a verifier is able to check the correctness of a combined block $m' = y_1m_1 + y_2m_2 \in Z_p$ without knowing block m_1 and m_2 by verifying:

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

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

$$\begin{aligned} & e(H_1(id_1)^{y_1}H_1(id_2)^{y_2}g_1^{m'}, g_2) \\ = \; & e(H_1(id_1)^{y_1}g_1^{y_1m_1}, g_2) \cdot e(H_1(id_2)^{y_2}g_1^{y_2m_2}, g_2) \\ = \; & e(\beta_1, g_2)^{y_1} \cdot e(\beta_2, g_2)^{y_2} \\ = \; & \prod_{i=1}^{d} e(\sigma_{1,i}, w_i)^{y_1} \cdot \prod_{i=1}^{d} e(\sigma_{2,i}, w_i)^{y_2} \\ = \; & \prod_{i=1}^{d} e(\sigma_{1,i}^{y_1} \cdot \sigma_{2,i}^{y_2}, w_i). \end{aligned}$$

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 verification.

Meanwhile, an adversary, who does not have any user's private key, cannot generate a valid ring signature $\boldsymbol{\sigma}'$ on an invalid block m' by linearly combining $\boldsymbol{\sigma}_1$ and $\boldsymbol{\sigma}_2$ with y_1 and y_2. Because if an element σ_i' in $\boldsymbol{\sigma}'$ is computed as $\sigma_i' = \sigma_{1,i}^{y_1} \cdot \sigma_{2,i}^{y_2}$, the whole ring signature $\boldsymbol{\sigma}' = (\sigma_1', ..., \sigma_d')$ cannot pass Equation (3) in **RingVerify**.

More specifically, if block m_1 and m_2 are signed by the same user, for example, user u_s, then σ'_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 ring signature $\boldsymbol{\sigma'} = (\sigma'_1, ..., \sigma'_d)$ is verified with the invalid block m' using Equation (3):

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

which means it always fails to pass the verification. Because $\beta_1^{y_1} \beta_2^{y_2} = H(id_1)^{y_1} H(id_2)^{y_2} g_1^{m'}$ is not equal to $\beta' = H(id')g_1^{m'}$.

If block m_1 and m_2 are signed by different users, for example, user u_s and user u_t, then σ'_s and σ'_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 ring signature $\boldsymbol{\sigma'} = (\sigma'_1, ..., \sigma'_d)$ is verified with the invalid block m' using Equation (3):

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

which means it always fails to pass the verification. Therefore, an adversary cannot output valid ring signatures on invalid blocks by linearly combining existing signatures, which indicates that HARS is non-malleable. Because HARS is not only blockless verifiable and but also non-malleable, it is a homomorphic authenticable signature scheme. □

Now, following the theorem in [5], we show that a verifier cannot distinguish the identity of the signer among a group of users under HARS.

***Theorem** 5: 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 $\boldsymbol{\sigma}$ is a ring signature generated with user u_s's private key $\boldsymbol{sk}_s$.*

Proof: For any $h \in G_1$, and any s, $1 \leq s \leq d$, the distribution $\{g_1^{a_1}, ..., g_1^{a_d} : a_i \overset{R}{\leftarrow} Z_p$ for $i \neq s$, a_s chosen such that $\prod_{i=1}^{d} g_1^{a_i} = h\}$ is identical to the distribution $\{g_1^{a_1}, ..., g_1^{a_d} : \prod_{i=1}^{d} g_1^{a_i} = h\}$. Therefore, given $\boldsymbol{\sigma} = (\sigma_1, ..., \sigma_d)$, the probability algorithm $\mathcal{A}$ distinguishes σ_s, which indicates the identity of the signer, is at most $1/d$. Details of the proof about identity privacy can be found in [5]. □

5 Privacy-Preserving Public Auditing for Shared Data in the Cloud

5.1 Overview

Using HARS and its properties we established in the previous section, we now construct Oruta, our privacy-preserving public auditing mechanism for shared data in the cloud. With Oruta, the TPA can verify the integrity of shared data for a group of users without retrieving the entire data. Meanwhile, the identity of the signer on each block in shared data is kept private from the TPA 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 Z_p and its ring signature contains d elements of G_1, where 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 is very frustrating for users, because cloud service providers, such as Amazon, will charge users based on the storage space they used. To reduce the storage for ring signatures and still allow the TPA to audit shared data efficiently, we exploit an aggregated approach from [6]. Specifically, we aggregate a block $\boldsymbol{m}_j = (m_{j,1}, ..., m_{j,k}) \in Z_p^k$ in shared data as $\prod_{l=1}^{k} \eta^{m_{j,l}}$ instead of computing g_1^m in Equation (1), where $\eta_1, ..., \eta_k$ are random values of G_1. With the aggregation, 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 [7]. 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 will be increasing with an increase of k.

5.3 Support Dynamic Operations

To enable each user in the group to easily modify data in the cloud and share the latest version of data with the rest of the group, Oruta should also support dynamic operations on shared data. An dynamic operation includes an insert, delete or update operation on a single block. 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, are not suitable for supporting dynamic operations on shared data. 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 Figure 3 and 4), and the changes of these indices require users to re-compute the signatures of these blocks, even though the content of these blocks are not modified.

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

Insert

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

Fig. 3. After inserting block $\boldsymbol{m}'_2$, all the indices 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	$\boldsymbol{m}_n$

Delete

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

Fig. 4. After deleting block $\boldsymbol{m}_2$, all the indices after block $\boldsymbol{m}_1$ are changed

By utilizing index hash tables [7], 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 [7], in our mechanism, an identifier from the index hash table is described as $id_j = \{v_j, r_j\}$, where v_j is the virtual index of block $\boldsymbol{m}_j$, and r_j is a random generated by a collision-resistance hash function $H_2 : \{0,1\}^* \rightarrow Z_q$ with $r_j = H_2(\boldsymbol{m}_j || v_j)$. Here, q is a much smaller prime than p. The collision-resistance of H_2 ensures that each block has a unique identifier. 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 $\boldsymbol{m}_j$ is computed as $v_j = j \cdot \delta$, where δ 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 $v'_j = (v_{j-1} + v_j)/2$. Clearly, if block $\boldsymbol{m}_j$ and block $\boldsymbol{m}_{j+1}$ are both originally created by the original user, the maximal number of inserted blocks that is allowed between block $\boldsymbol{m}_j$ and block $\boldsymbol{m}_{j+1}$ is δ. The original user can estimate and choose a proper value of δ based on the original size of shared data, the number of users in the group, the subject of content in shared data and so on, Generally, we believe $\delta = 10,000$ or $100,000$ is good enough for the maximal insert blocks between two blocks, which are originally created by the original user. Examples of different dynamic operations on shared data with index hash tables are described in Figure 5 and 6.

5.4 Construction of Oruta

Now, we present the details of our public auditing mechanism, Oruta. It includes five algorithms: **KeyGen**, **SigGen**, **Modify**, **ProofGen** and **ProofVerify**. In **KeyGen**, users generate their own public/private key pairs.

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

Insert

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

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

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

Update

Delete

Index	Block	V	R
1	$\boldsymbol{m}'_1$	δ	r'_1
2	$\boldsymbol{m}_2$	2δ	r_2
3	$\boldsymbol{m}_4$	4δ	r_4
4	$\boldsymbol{m}_5$	5δ	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.

In **SigGen**, a user (either the original user or a group user) is able to compute ring signatures on blocks in shared data. 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 the TPA and the cloud server together to generate a proof of possession of shared data. In **ProofVerify**, the TPA verifies the proof and sends an auditing report to the user.

Note that 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. Before the original user outsources shared data to the cloud, she decides all the group members, and computes all the initial ring signatures of all the blocks in shared data with her private key and all the group members' public keys. After shared data is stored in the cloud, when a group member modifies a block in shared data, this group member also needs to compute a new ring signature on the modified block.

Scheme Details. Let G_1, G_2 and G_T be multiplicative cyclic groups of order p, g_1 and g_2 be generators of groups G_1, G_2, respectively. Let $e : G_1 \times G_2 \rightarrow G_T$ be a bilinear map, and $\psi : G_2 \rightarrow G_1$ be a computable isomorphism with $\psi(g_2) = g_1$. There are three hash functions H_1: $\{0,1\}^* \rightarrow G_1$, $H_2 : \{0,1\}^* \rightarrow Z_q$ and $h : G_1 \rightarrow Z_p$. The global parameters are $(e, \psi, p, q, G_1, G_2, G_T, g_1, g_2, H_1, H_2, h)$. The total number of users in the group is d. Let U denote the group that includes all the d users.

Shared data M is divided into n blocks, and each block $\boldsymbol{m}_j$ is further divided into k elements in Z_p. Therefore,

shared data M can be described as a $n \times k$ matrix:

$$M = \begin{pmatrix} \boldsymbol{m}_1 \\ \vdots \\ \boldsymbol{m}_n \end{pmatrix} = \begin{pmatrix} m_{1,1} & \dots & m_{1,k} \\ \vdots & \ddots & \vdots \\ m_{n,1} & \dots & m_{n,k} \end{pmatrix} \in Z_p^{n \times k}.$$

KeyGen. For user u_i in the group U, she randomly picks $x_i \in Z_p$ and computes $w_i = g_2^{x_i}$. The user u_i's public key is $\mathbf{pk}_i = w_i$ and 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 η_l are random elements of 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 the ring signature of this block as follows:

1) She first aggregates block $\boldsymbol{m}_j$ with the public aggregate key **pak**, and computes

$$\beta_j = H_1(id_j) \prod_{l=1}^{k} \eta_l^{m_{j,l}} \in G_1. \quad (4)$$

2) After computing β_j, user u_s randomly chooses $a_{j,i} \in Z_p$ and sets $\sigma_{j,i} = g_1^{a_{j,i}}$, for all $i \neq s$. Then she calculates

$$\sigma_{j,s} = \left(\frac{\beta_j}{\psi(\prod_{i \neq s} w_i^{a_{j,i}})} \right)^{1/x_s} \in G_1. \quad (5)$$

The ring signature of block $\boldsymbol{m}_j$ is $\boldsymbol{\sigma}_j = (\sigma_{j,1}, ..., \sigma_{j,d}) \in G_1^d$.

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. She computes the new identifier of the inserted block $\boldsymbol{m}'_j$ as $id'_j = \{v'_j, r'_j\}$. The virtual index $v'_j = (v_{j-1} + v_j)/2$, and $r'_j = H_2(\boldsymbol{m}'_j || v'_j)$. For the rest of blocks, the identifiers of these blocks are not changed (as explained in Figure 5). 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. 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 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 user first sends an auditing request to the TPA. After receiving an auditing request, the TPA generates an auditing message [2] as follows:

1) The TPA randomly picks a c-element subset $\mathcal{J}$ of set $[1, n]$ to locate the c selected blocks that will be checked in this auditing process, where n is total number of blocks in shared data.
2) For $j \in \mathcal{J}$, the TPA generates a random value $y_j \in Z_q$.

Then, the TPA sends an auditing message $\{(j, y_j)\}_{j \in \mathcal{J}}$ to the cloud server (as illustrated in Fig. 7).

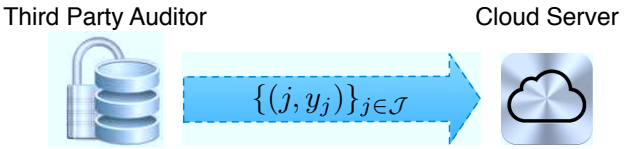

Fig. 7. The TPA sends an auditing message to the cloud server.

After receiving an auditing message $\{(j, y_j)\}_{j \in \mathcal{J}}$, the cloud server generates a proof of possession of selected blocks with the public aggregate key **pak**. More specifically:

1) The cloud server chooses a random element $r_l \in Z_q$, and calculates $\lambda_l = \eta_l^{r_l} \in G_1$, for $l \in [1, k]$.
2) To hide the linear combination of selected blocks using random masking, the cloud server computes $\mu_l = \sum_{j \in \mathcal{J}} y_j m_{j,l} + r_l h(\lambda_l) \in Z_p$, for $l \in [1, k]$.
3) The cloud server aggregates signatures as $\phi_i = \prod_{j \in \mathcal{J}} \sigma_{j,i}^{y_j}$, for $i \in [1, d]$.

After the computation, the cloud server sends an auditing proof $\{\boldsymbol{\lambda}, \boldsymbol{\mu}, \boldsymbol{\phi}, \{id_j\}_{j \in \mathcal{J}}\}$ to the TPA, where $\boldsymbol{\lambda} = (\lambda_1, ..., \lambda_k)$, $\boldsymbol{\mu} = (\mu_1, ..., \mu_k)$ and $\boldsymbol{\phi} = (\phi_1, ..., \phi_d)$ (as shown in Fig. 8).

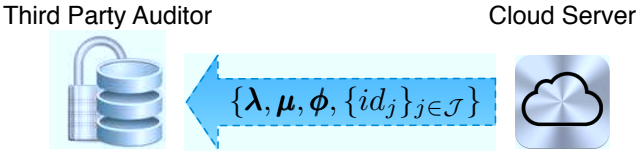

Fig. 8. The cloud server sends an auditing proof to the TPA.

ProofVerify. With an auditing proof $\{\boldsymbol{\lambda}, \boldsymbol{\mu}, \boldsymbol{\phi}, \{id_j\}_{j \in \mathcal{J}}\}$, an auditing message $\{(j, y_j)\}_{j \in \mathcal{J}}$, 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 TPA verifies the correctness of this proof by checking the following

equation:

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

If the above equation holds, then the TPA believes that the blocks in shared data are all correct, and sends a positive auditing report to the user. Otherwise, it sends a negative one.

Discussion. Based on the properties of bilinear maps, we can further improve the efficiency of verification by computing $d+2$ pairing operations in verification instead of computing $d+3$ pairing operations with Equation (6). Specifically, Equation (6) can also be described as

$$e(\prod_{j\in\mathcal{J}} H_1(id_j)^{y_j}\cdot\prod_{l=1}^{k}\eta_l^{\mu_l}\cdot(\prod_{l=1}^{k}\lambda_l^{h(\lambda_l)})^{-1}, g_2) \stackrel{?}{=} \prod_{i=1}^{d} e(\phi_i, w_i). \quad (7)$$

In the construction of Oruta, we leverage random masking [3] to support identity privacy. 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, such as attribute-based encryption (ABE) [8], [12]. With sampling strategies [2], the TPA can detect any corrupted block in shared data with a high probability by only choosing a subset of all blocks in each auditing task. Previous work [2] has already proved that, if the total number of blocks in shared data is $n = 1,000,000$ and 1% of all the blocks are lost or removed, the TPA can detect these corrupted blocks with a probability greater than 99% by choosing 460 random blocks.

5.5 Security Analysis of Oruta

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

***Theorem** 6: During an auditing task, the TPA is able to correctly audit the integrity of shared data under Oruta.*

Proof: To prove the correctness of Oruta is equivalent of proving Equation (6) is correct. Based on properties of bilinear maps and Theorem 1, the right-hand side (RHS) of Equation (6) can be expanded as follows:

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

□

***Theorem** 7: For an untrusted cloud, it is computational infeasible to generate an invalid auditing proof that can pass the verification under Oruta.*

Proof: As proved in Theorem **??**, for an untrusted cloud, if co-CDH problem in G_1 and G_2 is hard, it is computational infeasible to compute a valid ring signature on an invalid block under HARS.

In Oruta, besides generating valid ring signatures on arbitrary blocks, if the untrusted cloud can win Game 1, it can generate an invalid auditing proof for corrupted shared data, and successfully pass the verification. We define Game 1 as follows:

Game 1: The TPA sends an auditing message $\{j, y_j\}_{j\in\mathcal{J}}$ to the cloud, the correct auditing proof should be $\{\boldsymbol{\lambda}, \boldsymbol{\mu}, \boldsymbol{\phi}, \{id_j\}_{j\in\mathcal{J}}\}$, which can pass the verification with Equation (6). The untrusted cloud generates an invalid proof as $\{\boldsymbol{\lambda}, \boldsymbol{\mu}', \boldsymbol{\phi}, \{id_j\}_{j\in\mathcal{J}}\}$, where $\boldsymbol{\mu}' = (\mu'_1, ..., \mu'_k)$. Define $\Delta\mu_l = \mu'_l - \mu_l$ for $1 \le l \le k$, and at least one element of $\{\Delta\mu_l\}_{1\le l\le k}$ is nonzero. If this invalid proof still pass the verification, then the untrusted cloud wins. Otherwise, it fails.

Now, we prove that, if the untrusted cloud can win Game 1, we can find a solution to the Discrete Logarithm problem, which contradicts to the fact. We first assume the untrusted cloud can win Game 1. Then, according to Equation (6), we have

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

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

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

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 elements g, $h \in G_1$, there exists $x \in Z_p$ that $g = h^x$ because G_1 is a cyclic group. Without loss of generality, given g, $h \in G_1$, each η_l is able to randomly and correctly generated by computing $\eta_l = g^{\xi_l} h^{\gamma_l} \in G_1$, where ξ_l and γ_l are random values of 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 discrete logarithm problem. More specifically, given g, $h^x \in G_1$, we can compute

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

unless the denominator 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 γ_l is random element of Z_p, therefore, the denominator is zero with probability of $1/p$, which is negligible. It means, if the untrusted cloud wins Game 1, we can find a solution to the Discrete Logarithm problem with a probability of $1 - 1/p$, which contradicts the fact that the Discrete Logarithm problem is hard in G_1.

Therefore, for an untrusted cloud, it is computational infeasible to win Game 1 and generate an invalid proof, which can pass the verification. □

Now, we show that the TPA is able to audit the integrity of shared data, but the identity of the signer on each block in shared data is not disclosed to the TPA.

***Theorem** 8: During an auditing task, the probability for the TPA to distinguish the identities of all the signers on the c selected blocks in shared data is at most $1/d^c$.*

Proof: With Theorem 5, we have, for any algorithm $\mathcal{A}$, the probability to reveal the signer on one block in shared data is $1/d$. Because the c selected blocks in an auditing task are signed independently, the total probability that the TPA can distinguish all the signers' identities on the c selected blocks in shared data is at most $1/d^c$. □

Let us reconsider the example in Sec. 1. With Oruta, the TPA 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 shared data. However, it cannot distinguish who is the signer on a single block (as shown in Fig. 9). Therefore, this third party cannot obtain private and sensitive information, such as who signs the most blocks in shared data or which block is frequently modified by different group members.

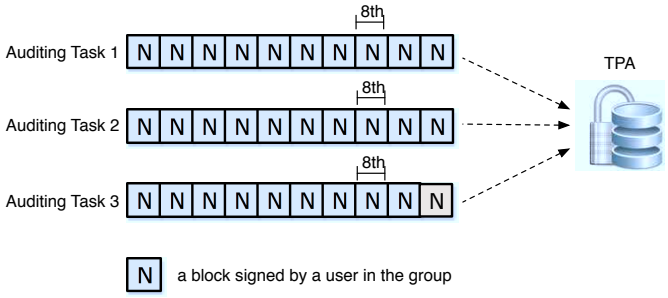

Fig. 9. Alice and Bob share a file in the cloud, and the TPA audit the integrity of shared data with Oruta.

Following a similar theorem in [3], we show that our scheme is also able to support data privacy.

***Theorem** 9: Given an auditing proof $= \{\boldsymbol{\lambda}, \boldsymbol{\mu}, \boldsymbol{\phi}, \{id_j\}_{j\mathcal{J}}\}$, it is computational infeasible for the TPA to reveal any private data in shared data under Oruta.*

Proof: If the combined element $\sum_{j\in\mathcal{J}} y_j m_{j,l}$, which is a linear combination of elements in blocks, is directly sent to the TPA, the TPA can learn the content of data by solving linear equations after collecting a sufficient number of linear combinations. To preserve private data from the TPA, the combined element is computed with random masking as $\mu_l = \sum_{j\in\mathcal{J}} y_j m_{j,l} + r_l h(\lambda_l)$. In order to still solve linear equations, the TPA must know the value of $r_l \in Z_p$. However, given $\eta_l \in G_1$ $\lambda_l = \eta_l^{r_l} \in G_1$, computing r_l is as hard as solving the Discrete Logarithm problem in G_1, which is computational infeasible. Therefore, give $\boldsymbol{\lambda}$ and $\boldsymbol{\mu}$, the TPA cannot directly obtain any linear combination of elements in blocks, and cannot further reveal any private data in shared data M by solving linear equations. □

5.6 Batch Auditing

With the usage of public auditing in the cloud, the TPA may receive amount of auditing requests from different users in a very short time. Unfortunately, allowing the TPA to verify the integrity of shared data for these users in several separate auditing tasks would be very inefficient. Therefore, with the properties of bilinear maps, we further extend Oruta to support batch auditing, which can improve the efficiency of verification on multiple auditing tasks.

More concretely, we assume there are B auditing tasks need to be operated, the shared data in all the B auditing tasks are denoted as $M_1, ..., M_B$ and the number of users sharing data M_b is described as d_b, where $1 \le b \le B$. To efficiently audit these shared data for different users in a single auditing task, the TPA sends an auditing message as $\{(j, y_j)\}_{j\in\mathcal{J}}$ to the cloud server. After receiving the auditing message, the cloud server generates 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 we presented in **ProofGen**, where $1 \le b \le B$, $1 \le l \le k$,

$1 \leq i \leq d_b$ and

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

Here $id_{b,j}$ is described as $id_{b,j} = \{f_b, v_j, r_j\}$, where f_b is the identifier of shared data M_b, e.g. the name of shared data M_b. Clearly, if two blocks are in the same shared data, these two blocks have the same identifier of shared data. As before, when a user modifies a single block in shared data M_b, the identifiers of other blocks in shared data M_b are not changed.

After the computation, the cloud server sends all the B auditing proofs together to the TPA. Finally, the TPA verifies the correctness of these B proofs simultaneously by checking the following equation with all the $\sum_{b=1}^{B} d_b$ users' public keys:

$$\begin{aligned} & e(\prod_{b=1}^{B} \left(\prod_{j \in \mathcal{J}} H(id_{b,j})^{y_j} \cdot \prod_{l=1}^{k} \eta_{b,l}^{\mu_{b,l}} \right), g_2) \\ \stackrel{?}{=} & \left(\prod_{b=1}^{B} \prod_{i=1}^{d_b} e(\phi_{b,i}, w_{b,i}) \right) \cdot e(\prod_{b=1}^{B} \prod_{l=1}^{k} \lambda_{b,l}^{h(\lambda_{b,l})}, g_2), \quad (8) \end{aligned}$$

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

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

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

If all the B auditing requests on B shared data are from the same group, the TPA can further improve the efficiency of batch auditing by verifying

$$\begin{aligned} & e(\prod_{b=1}^{B} \left(\prod_{j \in \mathcal{J}} H(id_{b,j})^{y_j} \cdot \prod_{l=1}^{k} \eta_{b,l}^{\mu_{b,l}} \right), g_2) \\ \stackrel{?}{=} & \left(\prod_{i=1}^{d} e(\prod_{b=1}^{B} \phi_{b,i}, w_i) \right) \cdot e(\prod_{b=1}^{B} \prod_{l=1}^{k} \lambda_{b,l}^{h(\lambda_{b,l})}, g_2). \quad (9) \end{aligned}$$

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 is only a small number of incorrect auditing proofs, we can utilize binary search [3] during batch auditing. More specifically, once the batch auditing of the B auditing proofs fails, the TPA divides the set of all the B auditing proofs into two subsets, which contains $B/2$ auditing proofs in each subset, and 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 TPA re-checks the correctness of auditing proofs in the each sub-subsets with batch auditing until all the incorrect auditing proofs are found. Clearly, when the number of incorrect auditing proofs increases, the efficiency of batch auditing will be reduced. Experimental results in Section 6 shows that, when less than 12% of auditing proofs among all the B auditing proofs are incorrect, batching auditing is still more efficient than verifying these auditing proofs one by one.

6 Performance

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

6.1 Computation Cost

The main cryptographic operations used in Oruta include multiplications, exponentiations, pairing and hashing operations. For simplicity, we omit additions in the following discussion, because they are much easier to be computed than the four types of operations mentioned above.

During auditing, the TPA first generates some random values to construct the auditing message, which only introduces a small cost in computation. Then, after receiving the auditing message, the cloud server needs to compute a proof $\{\boldsymbol{\lambda}, \boldsymbol{\mu}, \boldsymbol{\phi}, \{id_j\}_{j \in \mathcal{J}}\}$. The computation cost of calculating a proof is about $(k + dc)\mathrm{Exp}_{G_1} + dc\mathrm{Mul}_{G_1} + ck\mathrm{Mul}_{Z_p} + k\mathrm{Hash}_{Z_p}$, where Exp_{G_1} denotes the cost of computing one exponentiation in G_1, Mul_{G_1} denotes the cost of computing one multiplication in G_1, Mul_{Z_p} and Hash_{Z_p} respectively denote the cost of computing one multiplcation and one hashing operation in Z_p. To check the correctness of the proof $\{\boldsymbol{\lambda}, \boldsymbol{\mu}, \boldsymbol{\phi}, \{id_j\}_{j \in \mathcal{J}}\}$, the TPA verifies it based on Equation (6). The total cost of verifying the proof is $(2k + c)\mathrm{Exp}_{G_1} + (2k + c)\mathrm{Mul}_{G_1} + d\mathrm{Mul}_{G_T} + c\mathrm{Hash}_{G_1} + (d + 2)\mathrm{Pair}_{G_1,G_2}$. We use Pair_{G_1,G_2} to denote the cost of computing one pairing operation in G_1 and G_2.

6.2 Communication Cost

The communication cost of Oruta is mainly introduced by two factors: the auditing message and the auditing proof. For each auditing message $\{j, y_j\}_{j \in \mathcal{J}}$, the

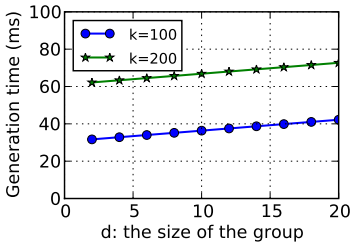

Fig. 10. Impact of d on signature generation time (ms).

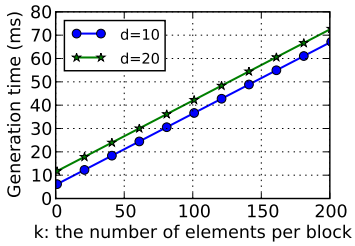

Fig. 11. Impact of k on signature generation time (ms).

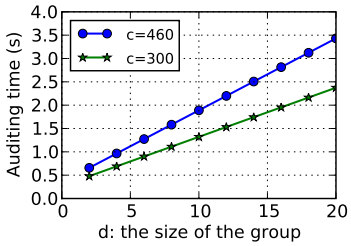

Fig. 12. Impact of d on auditing time (s), where $k = 100$.

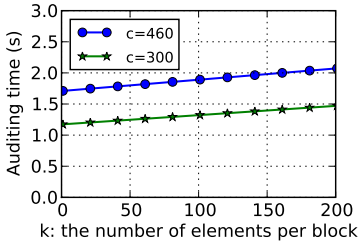

Fig. 13. Impact of k on auditing time (s), where $d = 10$.

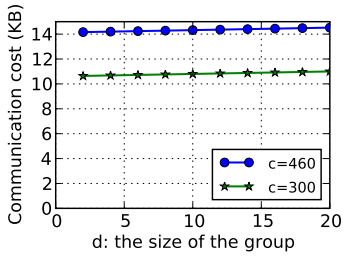

Fig. 14. Impact of d on communication cost (KB), where $k = 100$.

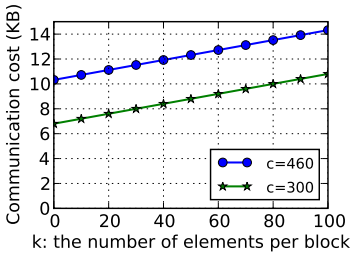

Fig. 15. Impact of k on communication cost (KB), where $d = 10$.

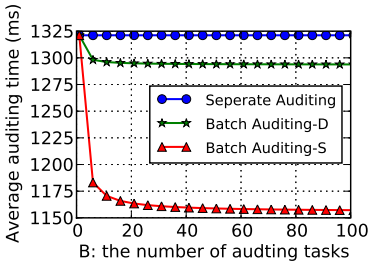

Fig. 16. Impact of B on the efficiency of batch auditing, where $k = 100$ and $d = 10$.

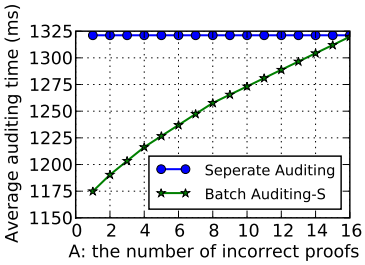

Fig. 17. Impact of A on the efficiency of batch auditing, where $B = 128$.

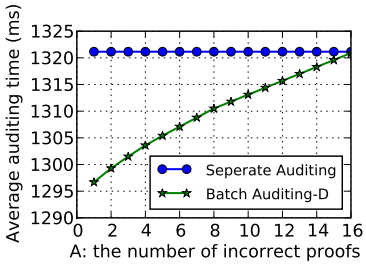

Fig. 18. Impact of A on the efficiency of batch auditing, where $B = 128$.

communication cost is $c(|q| + |n|)$ bits, where $|q|$ is the length of an element of Z_q and $|n|$ is the length of an index. Each auditing $= \{\boldsymbol{\lambda}, \boldsymbol{\mu}, \boldsymbol{\phi}, \{id_j\}_{j \in \mathcal{J}}\}$ contains $(k+d)$ elements of G_1, k elements of Z_p and c elememts of 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. To implement these complex cryptographic operations that we mentioned before, we utilize the GNU Multiple Precision Arithmetic (GMP)[2] library and Pairing Based Cryptography (PBC)[3] 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%, we set the number of selected blocks in an auditing task as $c = 460$ [2]. If only 300 blocks are selected, the detection probability is greater than 95%. 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.

6.3.1 *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

2. http://gmplib.org/
3. http://crypto.stanford.edu/pbc/

block. As illustrated in Fig. 10 and Fig. 11, 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.

6.3.2 Performance of Auditing

Based on our proceeding analysis, the auditing performance of Oruta under different detection probabilities is illustrated in Fig. 12–15, and Table 2. As shown in Fig. 12, 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 Fig. 14 and Fig. 15. Compare to the size of entire shared data, the communication cost that the TPA consumes in an auditing task is very small. It is clear in Table 2 that when maintaining a higher detection probability, the TPA needs to consume more computation and communication overhead to finish the auditing task. Specifically, when $c = 300$, it takes the TPA 1.32 seconds to audit the correctness of shared data, where the size of shared data is 2 GB; when $c = 460$, the TPA needs 1.94 seconds to verify the integrity of the same shared data.

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

6.3.3 Performance of Batch Auditing

As we discussed in Section 5, when there are multiple auditing tasks, the TPA can improve the efficiency of verification by performing batch auditing. In the following experiments, we choose $c = 300$, $k = 100$ and $d = 10$. We can see from Fig. 16 that, compare to verifying these auditing tasks one by one, if these B auditing tasks are from different groups, batching auditing can save 2.1% of the auditing time per auditing task on average. If these B auditing tasks are from the same group, batching auditing can save 12.6% of the average auditing time per auditing task.

Now we evaluate the performance of batch auditing when incorrect auditing proofs exist in 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, so that 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 (8) and (9), the extra computation cost in binary search is mainly introduced by extra pairing operations. As shown in Fig. 17, if all the 128 auditing proofs are from the same group, when the number of incorrect auditing proofs is less than 16 (12% of all the auditing proofs), batching auditing is still more efficient than separate auditing. Similarly, in Fig. 18, if all the auditing proofs are from different groups, 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), first proposed by Ateniese *et al.* [2], 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 verifiability or public auditing. Unfortunately, their mechanism is only suitable for auditing the integrity of static data. Juels and Kaliski [13] 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 [6] designed two improved POR schemes. The first scheme is built from BLS signatures, and the second one is based on pseudo-random functions.

To support dynamic operations on data, Ateniese *et al.* [14] 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.* utilized Merkle Hash Tree and BLS signatures [9] to support fully dynamic operations in a public auditing mechanism. Erway *et al.* [15] introduced dynamic provable data possession (DPDP) by using authenticated dictionaries,

which are based on rank information. Zhu *et al.* 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 for users. The public mechanism proposed by Wang *et al.* [3] is able to preserve users' confidential data from the TPA 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 [5].

Wang *et al.* [16] 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 operations on data, but also to identify misbehaved servers. To minimize communication overhead in the phase of data repair, Chen *et al.* [17] also introduced a mechanism for auditing the correctness of data with the multi-server scenario, where these data are encoded by network coding instead of using erasure codes. More recently, Cao *et al.* [18] constructed an LT codes-based secure and reliable cloud storage mechanism. Compare to previous work [16], [17], this mechanism can avoid high decoding computation cost for data users and save computation resource for online data owners during data repair.

To prevent special attacks exist in remote data storage system with *deduplication*, Halevi *et al.* [19] introduced the notation of proofs-of-ownership (POWs), which allows a client to prove to a server that she actually holds a data file, rather than just some hash values of the data file. Zheng *et al.* [20] further discussed that POW and PDP can co-exist under the same framework.

Recently, Franz *et al.* [21] proposed an oblivious outsourced storage scheme based on Oblivious RAM techniques, which is able to hide users' access patterns on outsourced data from an untrusted cloud. Vimercati *et al.* [22] utilize shuffle index structure to protect users' access patterns on outsourced data.

8 Conclusion

In this paper, we propose Oruta, the first privacy-preserving public auditing mechanism for shared data in the cloud. We utilize ring signatures to construct homomorphic authenticators, so the TPA is able to audit the integrity of shared data, yet cannot distinguish who is the signer on each block, which can achieve identity privacy. To improve the efficiency of verification for multiple auditing tasks, we further extend our mechanism to support batch auditing. An interesting problem in our future work is how to efficiently audit the integrity of shared data with dynamic groups while still preserving the identity of the signer on each block from the third party auditor.

References

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

[2] G. Ateniese, R. Burns, R. Curtmola, J. Herring, L. Kissner, Z. Peterson, and D. Song, "Provable Data Possession at Untrusted Stores," in *Proc. ACM Conference on Computer and Communications Security (CCS)*, 2007, pp. 598–610.

[3] C. Wang, Q. Wang, K. Ren, and W. Lou, "Privacy-Preserving Public Auditing for Data Storage Security in Cloud Computing," in *Proc. IEEE International Conference on Computer Communications (INFOCOM)*, 2010, pp. 525–533.

[4] R. L. Rivest, A. Shamir, and Y. Tauman, "How to Leak a Secret," in *Proc. International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT)*. Springer-Verlag, 2001, pp. 552–565.

[5] D. Boneh, C. Gentry, B. Lynn, and H. Shacham, "Aggregate and Verifiably Encrypted Signatures from Bilinear Maps," in *Proc. International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*. Springer-Verlag, 2003, pp. 416–432.

[6] H. Shacham and B. Waters, "Compact Proofs of Retrievability," in *Proc. International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT)*. Springer-Verlag, 2008, pp. 90–107.

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

[8] S. Yu, C. Wang, K. Ren, and W. Lou, "Achieving Secure, Scalable, and Fine-grained Data Access Control in Cloud Computing," in *Proc. IEEE International Conference on Computer Communications (INFOCOM)*, 2010, pp. 534–542.

[9] D. Boneh, B. Lynn, and H. Shacham, "Short Signature from the Weil Pairing," in *Proc. International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT)*. Springer-Verlag, 2001, pp. 514–532.

[10] D. Boneh and D. M. Freeman, "Homomorphic Signatures for Polynomial Functions," in *Proc. International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*. Springer-Verlag, 2011, pp. 149–168.

[11] A. L. Ferrara, M. Green, S. Hohenberger, and M. Ø. Pedersen, "Practical Short Signature Batch Verification," in *Proc. RSA Conference, the Cryptographers' Track (CT-RSA)*. Springer-Verlag, 2009, pp. 309–324.

[12] V. Goyal, O. Pandey, A. Sahai, and B. Waters, "Attribute-Based Encryption for Fine-Grained Access Control of Encrypted Data," in *Proc. ACM Conference on Computer and Communications Security (CCS)*, 2006, pp. 89–98.

[13] A. Juels and B. S. Kaliski, "PORs: Proofs pf Retrievability for Large Files," in *Proc. ACM Conference on Computer and Communications Security (CCS)*, 2007, pp. 584–597.

[14] G. Ateniese, R. D. Pietro, L. V. Mancini, and G. Tsudik, "Scalable and Efficient Provable Data Possession," in *Proc. International Conference on Security and Privacy in Communication Networks (SecureComm)*, 2008.

[15] C. Erway, A. Kupcu, C. Papamanthou, and R. Tamassia, "Dynamic Provable Data Possession," in *Proc. ACM Conference on Computer and Communications Security (CCS)*, 2009, pp. 213–222.

[16] C. Wang, Q. Wang, K. Ren, and W. Lou, "Ensuring Data Storage Security in Cloud Computing," in *Proc. IEEE/ACM International Workshop on Quality of Service (IWQoS)*, 2009, pp. 1–9.

[17] B. Chen, R. Curtmola, G. Ateniese, and R. Burns, "Remote Data Checking for Network Coding-based Distributed Stroage Systems," in *Proc. ACM Cloud Computing Security Workshop (CCSW)*, 2010, pp. 31–42.

[18] N. Cao, S. Yu, Z. Yang, W. Lou, and Y. T. Hou, "LT Codes-based Secure and Reliable Cloud Storage Service," in *Proc. IEEE International Conference on Computer Communications (INFOCOM)*, 2012.

[19] S. Halevi, D. Harnik, B. Pinkas, and A. Shulman-Peleg, "Proofs of Ownership in Remote Storage Systems," in *Proc. ACM Conference on Computer and Communications Security (CCS)*, 2011, pp. 491–500.

[20] Q. Zheng and S. Xu, "Secure and Efficient Proof of Storage with Deduplication," in *Proc. ACM Conference on Data and Application Security and Privacy (CODASPY)*, 2012.

[21] M. Franz, P. Williams, B. Carbunar, S. Katzenbeisser, and R. Sion, "Oblivious Outsourced Storage with Delegation," in *Proc. Financial Cryptography and Data Security Conference (FC)*, 2011, pp. 127–140.

[22] S. D. C. di Vimercati, S. Foresti, S. Paraboschi, G. Pelosi, and P. Samarati, "Efficient and Private Access to Outsourced Data,"

in *Proc. IEEE International Conference on Distributed Computing Systems (ICDCS)*, 2011, pp. 710–719.