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

Supraja .M
M.tech.,(CSE).
Sri Padmavati Mahila University
Tirupathi, India

T. Sudha(HOD)
Sri Padmavati Mahila University
Tirupathi, India

N. Padmaja (Guide)
Sri Padmavati Mahila University
Tirupathi, India

***Abstract*—With cloud storage services, it is common place 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.**

***keywords*—Public auditing, privacy-preserving,shared data, cloud computing.**

1 INTRODUCTION

Cloud service providers manage an enterprise-class infrastructure that offers a scalable, secure and re-liable 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.

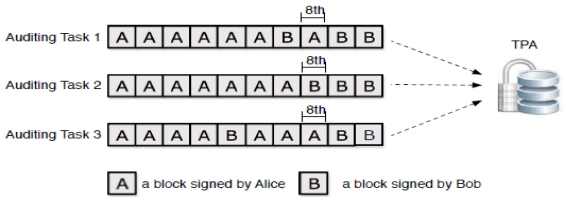

Fig. 1. Alice and Bob share a file in the cloud.

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. 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.

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

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.

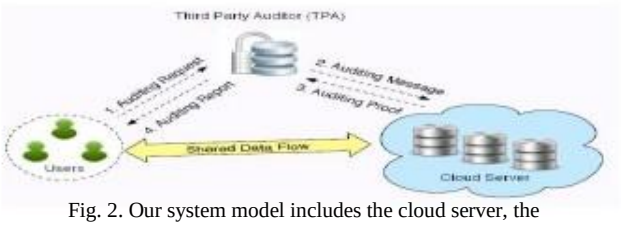

Fig. 2. Our system model includes the cloud server, the third party auditor and users.

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.

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 Au-diting:** 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 re-lated to bilinear maps.

1) G1, G2 and GT are three multiplicative cyclic groups of prime order p;
2) g1 is a generator of G1, and g2 is a generator of G2;
3) ψ is a computable isomorphism from G2 to G1,

with $\psi(g2) = g1$;

4) e is a bilinear map e: G1 × G2 → GT with the following properties:

Computability: there exists an efficiently computable algorithm for computing the map.

Bilinearity: for all u ∈ G1, v ∈ G2 and a, b ∈ Zp, e(ua, vb) = e(u, v)ab.

Non-degeneracy: $e(g1, g2) \neq 1$.

These properties further imply two additional properties: (1) for any u1, u2 ∈ G1 and v ∈ G2, e(u1 · u2, v) = e(u1, v) · e(u2, v); (2) for any u, v ∈ G2, $e(\psi(u), v) = e(\psi(v), u)$.

3.2 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.

3.3 Homomorphic Authenticators

Homomorphic authenticators (also called homomor- phic 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 signa- tures), a homomorphic authenticable signature scheme, which denotes a homomorphic authenticator based on signatures, should also satisfy the following properties:

Let (pk, **sk)** denote the signer's public/private key pair, σ1 denote a signature on block m1 ∈ Zp, σ2 denote a signature on block m2 ∈ Zp.

• **Blockless verification:** Given σ1 and σ2, two ran-dom values α1, α2 ∈ Zp and a block m′ = α1m1 + α2m2 ∈ Zp, a verifier is able to check the correctness of block m′ without knowing block m1 and m2.

• **Non-malleability** Given σ1 and σ2, two random values α1, α2 ∈ Zp and a block m′ = α1m1 + α2m2 ∈ Zp, 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 σ1 and σ2.

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 homomor-phic authenticable signatures includes aggregate sig-natures [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], de-noted 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.**

KeyGen. For a user ui in the group U, she randomly picks xi ∈ Zp and computes wi = g2xi ∈ G2. Then, user ui's public key is **pk**i = wi and her private key is **sk**i = xi.

RingSign. Given all the d users' public keys (pk1, ..., **pk**d) = (w1, ..., wd), a block m ∈ Zp, the iden-tifier of this block id and the private key **sk**s for some s, user us randomly chooses ai ∈ Zp for all i ≠ s, where i ∈ [1, d], and let σi = g1ai . Then, she computes

$$\beta = H1(id)g1m \in G1 . \quad (1)$$

and sets

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

and the ring signature of block m is **σ** = (σ1, ..., σd) ∈ Gd1.

RingVerify. Given all the d users' public keys (pk1, ..., **pk**d) = (w1, ..., wd), a block m, an identifier id and a ring signature **σ** (σ1, ..., σd), a verifier first computes β = H1(id)g1m ∈ G1, and then checks

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

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 equiva-lent of proving Equation (3) is correct. Based on prop-erties 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 (pk1, ..., **pk**d) = (w1, ..., wd), 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* A *is a* (t′, ϙ′)-algorithm *that can generate a forgery of a ring signature on a group of users of size* d. *Then there exists a* (t, ϙ)-algorithm *that can solve the co-CDH problem with* t ≤ 2t′+2cG1 (qH +dqs+qs+d)+2cG2 d *and* ϙ ≥ (ϙ′/(e + eqs))2, *where* A *issues at most* qH *hash queries and at most* qs *ring- signing queries,* e = limqs→∞(1+ 1/qs)qs , *exponentiation and inversion on* G1 *take time* cG1 , *and exponentiation and inversion on* G2 *take time* cG2 .

Proof: The co-CDH problem can be solved by solving two random instances of the following problem: Given g1ab, g2a (and g1,g2), compute g1b. We shall construct an algorithm B that solves this problem. This problem is easy if a = 0. In what follows, we assume a ≠ 0.

5 PRIVACY-PRESERVINGPUBLIC 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 Zp and its ring signature contains d elements of G1, where G1 is a cyclic group with order p. It means a |p|-bit block requires a d × |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 **m**j =

(mj,1, ..., mj,k) ∈ Zpk in shared data as ΠI=1kηmj,l instead of computing gm in Equation(1), where η1 ,…., ηk are random values of G1. 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 beincreasing 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 opera-tion 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

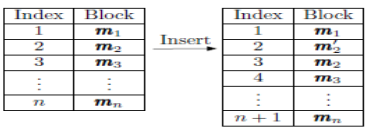

Fig. 3. After inserting block $\boldsymbol{m}_2'$, all the indices after block $\boldsymbol{m}_2'$ are changed

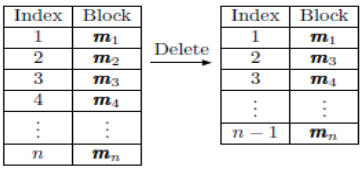

Fig. 4. After deleting block $\boldsymbol{m}_2$, all the indices after block $\boldsymbol{m}_1$ are changed

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 **Key-Gen,** users generate their own public/private key pairs.

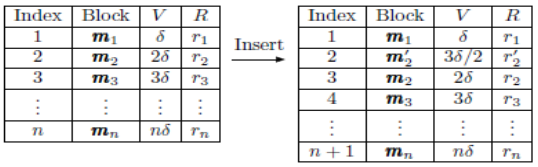

Fig. 5. Insert block $\boldsymbol{m}_2'$ into shared data using an index hash table as identifiers

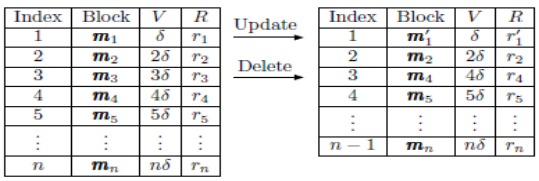

Fig. 6. Update block $\boldsymbol{m}_1$ and delete block $\boldsymbol{m}_3$ in shared data using an index hash table as identifiers.

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.

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 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 ϵ J , the TPA generates a random value yj ϵ Zq. Then, the TPA sends an auditing message {(j, yj)} j ϵ J to the cloud server (as illustrated inFig. 7).

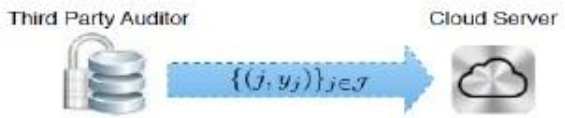

Fig. 7. The TPA sends an auditing message to the cloud server.

After receiving an auditing message {(j, yj)} j ϵ 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 rl ϵ Zq, and calculates λl = ηlrl ∈ G1, for l ∈ [1, k].

2) To hide the linear combination of selected blocks using random masking, the cloud server computes

$$\mu_l = \sum_{j\in J} y_j m_{j,l} + r_l h(\lambda_l) \in Z_p, \text{ for } l \in [1, k].$$

3) The cloud server aggregates signatures as

$$\phi_i = \pi \, j \epsilon J \, \sigma \, j,iyj, \text{ for } i \epsilon [1, d].$$

After the computation, the cloud server sends an auditing proof {λ,μ,φ, {idj }j∈J } to the TPA, where (λ1, ..., λk), **μ** = (μ1, ..., μk) and **φ** = (φ1, ..., φd) (as shown in Fig. 8).

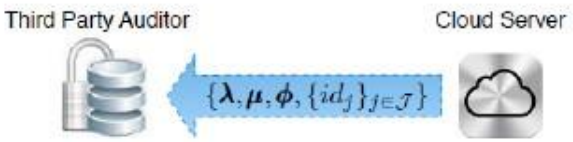

Fig. 8. The cloud server sends an auditing proof to the TPA.

ProofVerify. With an auditing proof {λ,μ,φ, {idj }j∈J, an auditing message {(j, yj)}j∈J , public aggregate key **pak** = (η1, ..., ηk), and all the group members' public keys (pk1, ..., **pk**d) = (w1, ..., wd), the TPA verifies the correctness of this proof by checking the following equation

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

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 bedescribed as

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

5.5Security Analysis of Oruta

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

***THEOREM** 3: 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:

$$\text{RHS} = \left(\prod_{i=1}^{d} e(\prod_{j\in J} \sigma_{j,i}^{y_j}, w_i)\right) \cdot e(\prod_{l=1}^{k} \lambda_l^{h(\lambda_l)}, g_2)$$

***THEOREM 4:** 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 G1 and G2 is hard, it is computational infeasible to compute a valid ring signa-ture on an invalid block under HARS.

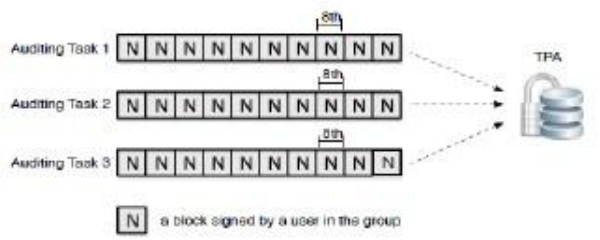

Fig. 9. Alice and Bob share a file in the cloud, and the TPA audit the integrity of shared data with Oruta.

Following a similar theorem in [2], we show that our scheme is also able to support data privacy.

5.6 Batch Auditing

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 M1, ..., MB and the number of users sharing data Mb is described as db, where 1 ≤ b ≤ B. To efficiently audit these shared data for different users in a single auditing task, the

TPA sends an auditing message as {(j, yj)}j∈J to the cloud server. After receiving the auditing message, the cloud server generates an auditing proof {λb,μb,φb, {idb,j }j∈J } for each shared data Mb as we presented in **ProofGen,** where 1 ≤ b ≤ B, 1 ≤ l ≤ k,

$1 \le i \le d_b$ 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 idb,j is described as idb,j = {fb, vj , rj },where fb is the identifier of shared data Mb, e.g. the name of shared data Mb. 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 Mb, the identifiers of other blocks in shared data Mb 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 ΣBb=1 db users' public keys:

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

where **pk**b,i = wb,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 correct-ness of batch auditing can be presented as follows:

$$\left(\prod_{b=1}^{B}\prod_{i=1}^{d_b} e(\phi_{b,i}, w_{b,i})\right) \cdot e(\prod_{b=1}^{B}\prod_{l=1}^{k} \lambda_{b,l}^{h(\lambda_{b,l})}, g_2)$$

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

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

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.

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 essage {j, yj}j2J , the communication cost is c(|q| + |n|) bits, where |q| is the length of an element of Zq and |n| is the length of an index. Each auditing = {λ,μ,ϕ, {idj}jϵJ } contains (k+d) elements of G1, k elements of Zp and c elememts of Zq, 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 Pair-ing 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.

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 twoimproved POR schemes. The first scheme is built from BLS signatures, and the second one is based on pseudo random functions.

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. Kon-winski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, and M. Za-haria, ―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. Peter-son, and D. Song, ―Provable Data Possession at Untrusted Stores,‖ in *Proc. ACMConferenceonComputerand 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 CryptologyandInformationSecurity (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. In-ternational 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 CryptologyandInformationSecurity (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 Con-ference, the Cryptographers' Track (CT-RSA).* Springer-Verlag, 2009, pp. 309–324.