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

Boyang Wang [†,††], Baochun Li [††] and Hui Li [†]
[†] State Key Laboratory of Integrated Services Networks, Xidian University, Xi'an, China
[††] Department of Electrical and Computer Engineering, University of Toronto, Toronto, Canada
Email:{bywang,bli}@eecg.toronto.edu, lihui@mail.xidian.edu.cn

***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 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**

I. 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 group, 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 more 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 in the group, this user needs to sign the new block using her private key. 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.

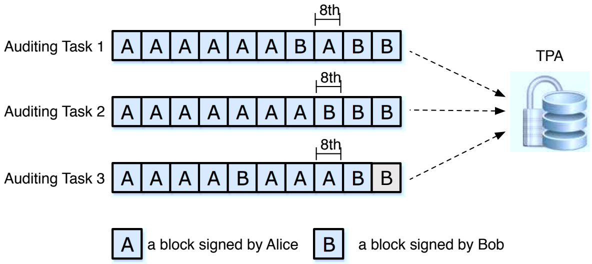


Fig. 1. Alice and Bob share a file in the cloud, and the TPA audit the integrity of data with existing mechanisms.

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 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 above example, the identities of signers on shared data may indicate which user in the group or block in shared data is a more 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

[1] Oruta: One Ring to Rule Them All.

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. Meanwhile, the identity of the signer on each block in shared data is kept private from the TPA. In addition, Oruta can continue to support data privacy and dynamic operations on data during public auditing. A high-level comparison between Oruta and existing mechanisms in the literature is shown in Table I. 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 I
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 Sec. II, we present the system model and threat model. In Sec. III, we briefly introduce cryptographic primitives used in Oruta. The detailed design and security analysis of Oruta are presented in Sec. IV and Sec. V. Sec. VI evaluates the performance of Oruta. Finally, we discuss related work in Sec. VII, and conclude this paper in Sec. VIII.

II. PROBLEM STATEMENT

A. 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. 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.

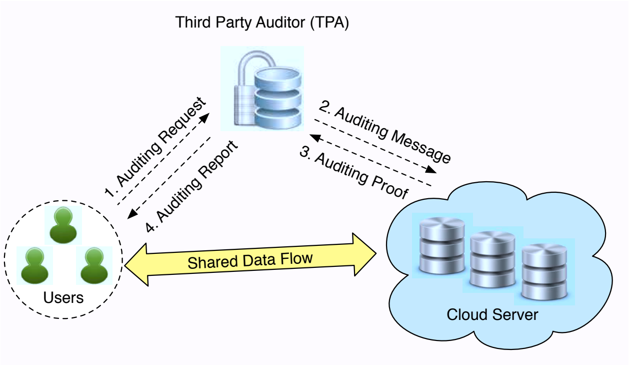


Fig. 2. Our system model includes the cloud server, the third party auditor and users.

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.

B. Threat Model

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) 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).

C. 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 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.

III. PRELIMINARIES

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

A. Bilinear Maps

Let G_1, G_2 and G_T be three multiplicative cyclic groups of prime order p, g_1 be a generator of G_1, and g_2 be a generator of G_2. A bilinear map e is a map e: $G_1 \times G_2 \rightarrow G_T$ with the following properties:

- **Computability**: there exists an efficiently computable algorithm for computing 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**: $e(g_1, g_2) \neq 1$.

B. 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.

C. 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.

D. Homomorphic Authenticators

Homomorphic authenticators (also called homomorphic verifiable tags) are basic tools to construct data auditing mechanisms [2], [3], [6]–[9]. 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 α_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 special 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 arbitrary blocks by combining existing signatures.

IV. Homomorphic Authenticable Ring Signatures

A. Overview

As we introduced in previous sections, we intend to utilize ring signatures to hide the identity of the signer on each block, so that private and sensitive information of the group is not disclosed to 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.

B. 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 $e : G_1 \times G_2 \rightarrow G_T$ be a bilinear map, and $\psi : G_2 \rightarrow G_1$ be a computable isomorphism with $\psi(g_2) = g_1$. There is a hash function H_1: $\{0,1\}^* \rightarrow G_1$. 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.

KeyGen. User u_i 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, user u_s 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 on 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

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

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.

C. 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:

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

■

***Theorem** 2: For an adversary, it is computational infeasible to forge a ring signature under HARS.*

Proof: Due to space limitations, we only provide the sketch of the proof in this paper. The full proof of this theorem can be found in our technical report [9].

By following the security model and the game defined in BGLS [5], we can prove that, if a (t', ϵ')-algorithm $\mathcal{A}$ 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 $\epsilon \geq (\epsilon'/(\hat{e} + \hat{e}q_s))^2$, where $\mathcal{A}$ issues at most q_H hash queries and at most q_s ring-signing queries, $\hat{e} = \lim_{q_s \to \infty}(1 + 1/q_s)^{q_s}$, exponentiation and inversion on G_1 take time c_{G_1}, and exponentiation and inversion on G_2 take time c_{G_2}. However, due to the assumption that the co-CDH problem is hard in G_1 and G_2, it is computational infeasible to find a (t', ϵ')-algorithm $\mathcal{A}$ that can generate a forgery of a ring signature under HARS. ■

Then, based on Theorem 1 and 2, we show that HARS is a homomorphic authenticable ring signature scheme.

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

Proof: To prove HARS is a homomorphic authenticable ring signature scheme, we first prove that HARS is able to support blockless verification, which we defined in Section III. 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 $m' = y_1 m_1 + y_2 m_2 \in Z_p$ without knowing block m_1 and m_2 by verifying:

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

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

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

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 the combined block m' by 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 combined 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 combined 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 combined blocks by 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. ■

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

Proof: For any $h \in G_1$, and any s, $1 \leq s \leq d$, the distribution $\{g_1^{a_1}, ..., g_1^{a_d} : a_i \xleftarrow{R} Z_p$ 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 $\mathcal{A}$ determines σ_s, which reveals the identity of the signer, is at most $1/d$. ■

V. Privacy-Preserving Public Auditing for Shared Data in the Cloud

A. Overview

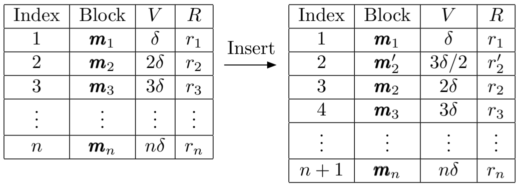
Using HARS and its properties we established in the previous section, we now construct Oruta, our privacy-preserving public auditing mechanism for shared data in the cloud. With Oruta, the TPA can verify the integrity of shared data for a group of users without retrieving the entire data. Meanwhile, the identity of the signer on each block in shared data is kept private from the TPA during the auditing.

B. 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. Generally, to obtain a smaller size of a ring signature than the size of a block, we choose $k > d$. As a trade-off, the communication cost of an auditing task will be increasing with an increase of k.

C. Support Dynamic Operations

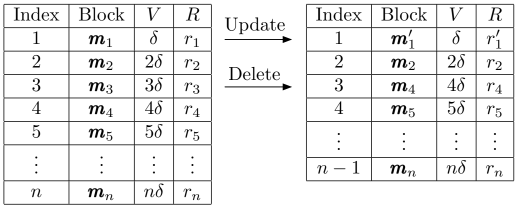
To enable each user in the group to easily modify data and share the latest version of data with the rest of the group, Oruta should also support dynamic operations on shared data. An dynamic operation indicates an insert, delete or update operation on a single block. However, since the computation of 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 (e.g. the index of block $\boldsymbol{m}_j$ is j), 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, 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	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

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

By utilizing index hash tables [8], 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 [8], 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. Examples of different dynamic operations on shared data with index hash tables are described in Figure 3 and 4.

D. Construction of Oruta

Now, we present the details of our public auditing mechanism, Oruta. It includes four algorithms: **KeyGen**, **SigGen**, **ProofGen** and **ProofVerify**. In **KeyGen**, users generate their own public/private key pairs. In **SigGen**, a user (either the original user or a group user) is able to compute ring signatures on blocks in shared data. **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.

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. User u_i randomly picks $x_i \in Z_p$ and computes $w_i = g_2^{x_i}$. 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

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

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

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

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

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.

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

1) 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) 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) 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 outputs an auditing proof $\{\boldsymbol{\lambda}, \boldsymbol{\mu}, \boldsymbol{\phi}, \{id_j\}_{j \in \mathcal{J}}\}$, and sends it to the TPA, where $\boldsymbol{\lambda} = (\lambda_1, ..., \lambda_k)$, $\boldsymbol{\mu} = (\mu_1, ..., \mu_k)$ and $\boldsymbol{\phi} = (\phi_1, ..., \phi_d)$.

ProofVerify. With an auditing message $\{(j, y_j)\}_{j \in \mathcal{J}}$, an auditing proof $\{\boldsymbol{\lambda}, \boldsymbol{\mu}, \boldsymbol{\phi}, \{id_j\}_{j \in \mathcal{J}}\}$, 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:

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

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.

E. Security Analysis of Oruta

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

***Theorem** 5: 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:

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

■

***Theorem** 6:* *For an untrusted cloud, it is computational infeasible to generate a forgery of an auditing proof under Oruta.*

Proof: Following the security model and the game defined in [6], we first define a game, named Game 1, as follows:

Game 1: The TPA sends an auditing message $\{j, y_j\}_{j\in\mathcal{J}}$ to the cloud, the auditing proof on the correct shared data M should be $\{\boldsymbol{\lambda}, \boldsymbol{\mu}, \boldsymbol{\phi}, \{id_j\}_{j\in\mathcal{J}}\}$, which should pass the verification with Equation (6). However, the untrusted cloud generates a proof on incorrect shared data M' as $\{\boldsymbol{\lambda}, \boldsymbol{\mu}', \boldsymbol{\phi}, \{id_j\}_{j\in\mathcal{J}}\}$, where $\boldsymbol{\mu}' = (\mu'_1, ..., \mu'_k)$ and $\mu'_l = \sum_{j\in\mathcal{J}} y_j m'_{j,l} + r_l h(\lambda_l) \in Z_p$, for $l \in [1, k]$. Define $\Delta\mu_l = \mu'_l - \mu_l$ for $1 \leq l \leq k$, and at least one element of $\{\Delta\mu_l\}_{1\leq l\leq k}$ is nonzero. If this proof still pass the verification, then the untrusted cloud wins. Otherwise, it fails.

If the untrusted cloud could win Game 1, we can find a solution to the Discrete Logarithm problem in G_1 with probability of $1-1/p$, which contradicts to the assumption 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 a forgery of an auditing proof on incorrect shared data. Due to space limitations, the full proof of this theorem can be found in our technical report [9]. ■

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** 7:* *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 4, 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 $1/d^c$. ■

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

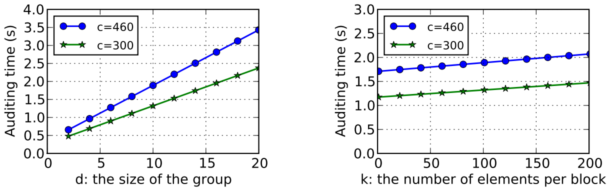
***Theorem** 8:* *Given an auditing proof $= \{\boldsymbol{\lambda}, \boldsymbol{\mu}, \boldsymbol{\phi}, \{id_j\}_{j\mathcal{J}}$ $\}$, it is computational infeasible for the TPA to reveal any private data in shared data under Oruta.*

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 r_l 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. ■

VI. Performance

We now evaluate the performance of Oruta. Due to space limitations, we only provide some experimental results of Oruta in this paper. Detailed analysis of computation and communication cost, and further experimental results can be found in our technical report [9].

In the following experiments,, we utilize the GNU Multiple Precision Arithmetic (GMP) library and Pairing Based Cryptography (PBC) library to simulate the cryptographic operations in Oruta, and all the experiments are tested on a 2.26GHz Linux system over $1,000$ times. We assume $|p| = 160$ bits, $|q| = 80$ bits, the number of blocks in shared data is $n = 1,000,000$. According to previous work [2], to keep the detection probability greater than 99%, we set the number of selected blocks in an auditing task as $c = 460$. If only 300 blocks are selected, the detection probability is greater than 95%.



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

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

Fig. 5. Performance of Auditing

1) Performance of Auditing: According to the presentation in Section V, the auditing time of an auditing task is determined by the number of users in the group, the number of elements in each block and the number of selected block in this auditing task. As shown in Fig. 5(a), when k is fixed, the auditing time of the entire shared data is linearly increasing with the size of the group. Similarly, when d is fixed in Fig. 5(b), the auditing time of the entire shared data is linearly increasing with the number of elements in each block. It is clear in Table II that Oruta can efficiently audit the integrity of shared data without downloading the entire data. Specifically, when the size of shared data is 2 GB and $c = 300$, the time of auditing the integrity of entire data is 1.32 seconds and the communication cost is only 10.95 KB. Compare to the total size of shared data, the communication cost of an auditing task is quite small. We can also see that, to maintain a higher detection probability, the TPA needs to consume more computation and communication overhead to finish the auditing task on shared data.

TABLE II
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

VII. RELATED WORK

Ateniese *et al.* [2] first proposed provable data possession (PDP), which allows a client to verify the integrity of her data stored in an untrusted server without retrieving the entire file. PDP is the first mechanism that provides public verifiability (also referred to as public auditing). However, it only supports static data. To improve the efficiency of verification, Ateniese *et al.* [10] constructed a scalable and efficient PDP using symmetric keys. This mechanism is able to support partially dynamic data operations. Unfortunately, it cannot support public verifiability and only offers each user a limited number of verification requests.

Juels and Kaliski [11] defined another similar model called proof of retrievability (POR), which is also able to check the correctness of data stored in an untrusted server. The original file is added with a set of randomly-valued blocks called *sentinels*. The user verifies the integrity of data by asking the server to return specific sentinel values. Shacham and Waters [6] designed two improved POR, which are built on pseudo-random functions and BLS signatures [12].

Wang *et al.* [7] leveraged the Merkle Hash Tree to construct a public auditing mechanism, which can support *fully dynamic data*. Erway *et al.* [13] also presented a fully dynamic PDP based on the rank-based authenticated dictionary. Zhu *et al.* [8] exploited index hash tables to support fully dynamic data during the public auditing process.

More recently, Wang *et al.* [3] first considered public auditing for cloud data with data privacy. In this mechanism, the third party auditor is able to check the integrity of cloud data but cannot obtain any private data. In addition, to operate multiple users' auditing tasks simultaneously, they also extended their mechanism to enable batch auditing by leveraging aggregate signatures [5]. Our recent work [14] is able to audit the integrity of shared data in the cloud for large groups. Unfortunately, it cannot support public auditing.

VIII. CONCLUSION

In this paper, we propose Oruta, the first privacy-preserving public auditing mechanism for shared data in the cloud. With Oruta, the TPA is able to efficiently audit the integrity of shared data, yet cannot distinguish who is the signer on each block, which can preserve identity privacy for users. 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.

ACKNOWLEDGEMENT

We are grateful to the anonymous reviewers for their helpful comments. This work is supported by the National Science and Technology Major Project (No. 2012ZX03002003), Fundamental Research Funds for the Central Universities (No. K50511010001), National 111 Program (No. B08038), Doctoral Foundation of Ministry of Education of China (No. 20100203110002) and Program for Changjiang Scholars and Innovative Research Team in University.

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, April 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 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 INFOCOM*, 2010, pp. 525–533.

[4] R. L. Rivest, A. Shamir, and Y. Tauman, "How to Leak a Secret," in *Proc. 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. EUROCRYPT*. Springer-Verlag, 2003, pp. 416–432.

[6] H. Shacham and B. Waters, "Compact Proofs of Retrievability," in *Proc. ASIACRYPT*. Springer-Verlag, 2008, pp. 90–107.

[7] Q. Wang, C. Wang, J. Li, K. Ren, and W. Lou, "Enabling Public Verifiability and Data Dynamic for Storage Security in Cloud Computing," in *Proc. European Symposium on Research in Computer Security*. Springer-Verlag, 2009, pp. 355–370.

[8] 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*, 2011, pp. 1550–1557.

[9] B. Wang, B. Li, and H. Li, "Oruta: Privacy-Preserving Public Auditing for Shared Data in the Cloud," University of Toronto, Tech. Rep., 2011. [Online]. Available: http://iqua.ece.toronto.edu/~bli/techreports/oruta.pdf

[10] G. Ateniese, R. D. Pietro, L. V. Mancini, and G. Tsudik, "Scalable and Efficient Provable Data Possession," in *Proc. ICST SecureComm*, 2008.

[11] A. Juels and B. S. Kaliski, "PORs: Proofs pf Retrievability for Large Files," in *Proc. ACM CCS*, 2007, pp. 584–597.

[12] D. Boneh, B. Lynn, and H. Shacham, "Short Signature from the Weil Pairing," in *Proc. ASIACRYPT*. Springer-Verlag, 2001, pp. 514–532.

[13] C. Erway, A. Kupcu, C. Papamanthou, and R. Tamassia, "Dynamic Provable Data Possession," in *Proc. ACM CCS*, 2009, pp. 213–222.

[14] B. Wang, B. Li, and H. Li, "Knox: Privacy-Preserving Auditing for Shared Data with Large Groups in the Cloud," in *Proc. ACNS*. Spring-Verlag, 2012.