Public Auditing for Shared Data with Efficient User Revocation in the Cloud

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

***Abstract*—With data services in the cloud, users can easily modify and share data as a group. To ensure data integrity can be audited publicly, users need to compute signatures on all the blocks in shared data. Different blocks are signed by different users due to data modifications performed by different users. For security reasons, once a user is revoked from the group, the blocks, which were previously signed by this revoked user must be re-signed by an existing user. The straightforward method, which allows an existing user to download the corresponding part of shared data and re-sign it during user revocation, is inefficient due to the large size of shared data in the cloud. In this paper, we propose a novel public auditing mechanism for the integrity of shared data with efficient user revocation in mind. By utilizing proxy re-signatures, we allow the cloud to re-sign blocks on behalf of existing users during user revocation, so that existing users do not need to download and re-sign blocks by themselves. In addition, a public verifier is always able to audit the integrity of shared data without retrieving the entire data from the cloud, even if some part of shared data has been re-signed by the cloud. Experimental results show that our mechanism can significantly improve the efficiency of user revocation.**

I. Introduction

With data storage and sharing services, such as Google Drive, provided by the cloud, people can easily work together as a group by sharing data with each other. More specifically, once a user creates shared data in the cloud, every user in the group is able to not only access and modify shared data, but also share the latest version of the shared data with the rest of the group. Although cloud providers promise a more secure and reliable environment to the users, the integrity of data in the cloud may still be compromised, due to the existence of hardware/software failures and human errors [1], [2].

To protect the integrity of data in an untrusted cloud, a number of mechanisms [2]–[9] have been proposed. In these mechanisms, a signature is attached to each block in data, and the integrity of data relies on the correctness of these signatures. One of the most significant and common features of these mechanisms is their ability to allow not only the data owner, but also a public verifier, such as a third party auditor (TPA), to check data integrity in the cloud without downloading the entire data, referred to as *public auditing*. Most of the previous works [2]–[7], [9] focus on auditing the integrity of *personal data*. Different from these works, our recent work [8] focuses on how to preserve identity privacy from the TPA when auditing the integrity of *shared data*. Unfortunately, none of the previous works, including our own, considers the efficiency of user revocation when auditing the correctness of shared data in the cloud.

With shared data, once a user modifies a block, she also needs to compute a new signature for the modified block. Due to the modifications from different users, different blocks are signed by different users. For security reasons, when a user leaves the group or misbehaves, this user must be revoked from the group. As a result, this revoked user should no longer be able to access and modify shared data, and the signatures generated by this revoked user are no longer valid to the group [10]. Therefore, although the content of shared data is not changed during user revocation, the blocks, which were previously signed by the revoked user, still need to be re-signed by an existing user in the group, so that, after the revocation, the integrity of the entire data can still be verified with the public keys of existing users only.

Since shared data is outsourced to the cloud and users no longer store it on local devices, the straightforward method to re-compute these signatures during user revocation (as shown in Fig. 1) is to allow an existing user to first download the blocks signed by the revoked user, verify the correctness of these blocks, then re-sign these blocks, and finally upload the new signatures to the cloud. However, this straightforward method may cost the existing user a huge amount of communication and computation resources by downloading and verifying blocks, and by re-computing and uploading signatures, especially when the number of re-signed blocks is quite large or the membership of the group is frequently changing. To make matters worse, the size of shared data in the cloud is generally large, which further prevents existing users from downloading and re-signing data efficiently.

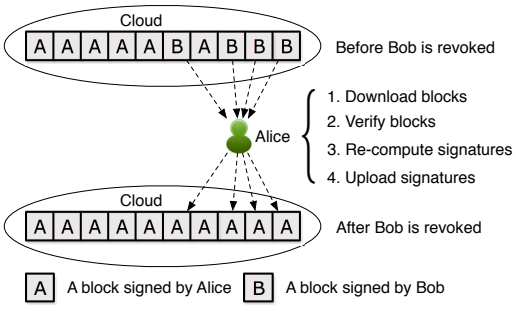

Fig. 1. Alice and Bob share data in the cloud. When Bob is revoked, Alice re-signs the blocks that were previously signed by Bob with her private key.

Clearly, if the cloud could possess each user's private key, it

can easily finish the re-signing task for existing users without asking them to download and re-sign blocks. However, since the cloud is not in the same trusted domain with each user in the group, outsourcing every user's private key to the cloud would introduce significant security issues. Another important problem we need to consider is that the re-computation of any signature during user revocation should not affect the most attractive property of public auditing — auditing data integrity publicly without retrieving the entire data. Therefore, how to efficiently reduce the significant burden to existing users introduced by user revocation, and still allow a public verifier to check the integrity of shared data without downloading the entire data from the cloud, is a challenging task.

In this paper, we propose a novel public auditing mechanism for the integrity of shared data with efficient user revocation in an untrusted cloud. In our mechanism, by utilizing the idea of proxy re-signatures [11], once a user in the group is revoked, the cloud is able to re-sign the blocks, which were signed by the revoked user, with a *re-signing key* (as presented in Fig. 2). As a result, the efficiency of user revocation can be significantly improved, and computation and communication resources of existing users can be easily saved. Meanwhile, the cloud, who is not in the same trusted domain with each user, is only able to convert a signature of the revoked user into a signature of an existing user on the same block, but it cannot sign arbitrary blocks on behalf of either the revoked user or an existing user. By designing a new proxy re-signature scheme with nice properties, which traditional proxy re-signatures do no have, even after the cloud re-signs any block, a public verifier is always able to check the integrity of shared data without retrieving the entire data from the cloud.

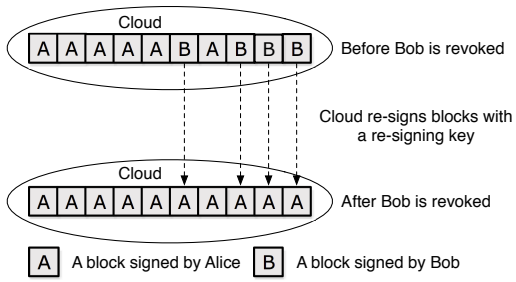

Fig. 2. When Bob is revoked, the cloud re-signs the blocks that were previously signed by Bob with a re-signing key.

The remainder of this paper is organized as follows: In Sec. II, we present the system model, threat model and design goals. Then, we introduce several cryptographic primitives in Sec. III. Detailed design and security analysis of our mechanism are presented in Sec. IV and Sec. V. We evaluate the performance of our mechanism in Sec. VI. Finally, we briefly discuss related work in Sec. VII, and conclude this paper in Sec. VIII.

II. Problem Statement

In this section, we describe the system and threat model of this paper, and illustrate the design goals of our public auditing mechanism.

A. System and Threat Model

In this paper, the system model includes three entities: the cloud, the third party auditor (TPA), and users who share data as a group (as illustrated in Fig. 3). The cloud offers data storage and sharing services to users. The TPA is able to publicly audit the integrity of shared data in the cloud for users. In a group, there is one original user and a number of group users. The original user is the original owner of data. This original user creates and shares data with other users in the group through the cloud. Both the original user and group users are able to access, download and modify shared data. Shared data is further divided into a number of blocks. A user can modify a block in shared data by performing an insert, delete or update operation on the block.

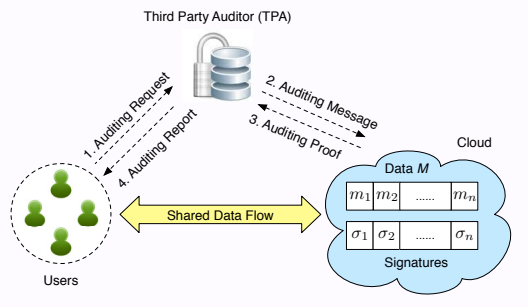

Fig. 3. The system model includes the cloud, the TPA, and users.

Generally, the integrity of shared data is threatened by three factors. First, the cloud service provider may inadvertently pollute shared data due to hardware/software failures and human errors. Second, an external adversary may try to corrupt shared data in the cloud, and prevent users from using shared data correctly. Third, a revoked user, who no longer has the right as existing users, may try to illegally modify shared data. Considering these threats, users do not fully trust the cloud with the integrity of shared data.

To protect the integrity of shared data, each block in shared data is attached with a signature, which is computed by one of the users in the group. When shared data is initially created by the original user in the cloud, all the signatures on shared data are computed by the original user. After that, once a user modifies a block, this user also needs to sign the modified block with his/her own private key. By sharing data among a group of users, different blocks may be signed by different users due to modifications from different users.

When a user in the group leaves or misbehaves, the group needs to revoke this user. Generally, as the creator of shared data, the original user acts as the group manager and is able to revoke users on behalf of the group. Once a user is revoked, the signatures computed by this revoked user become invalid to the group, and the blocks that were previously signed by this revoked user need to be re-signed by an existing user, so that the correctness of the entire data can still be verified with the public keys of existing users only.

Note that allowing every user in the group to share a common *group private key* and sign each block with it, is also a possible way to protect the integrity of shared data. However, when a user is revoked from the group, a new group private key needs to be securely distributed to every existing user and all the blocks in the shared data have to be re-signed with the new private key, which increases the complexity of key management and affects the efficiency of user revocation.

B. Design Goals

To correctly verify the integrity of shared data with efficient user revocation, our public auditing mechanism should achieve the following properties: (1) ***Correctness***: The TPA is able to correctly check the integrity of shared data. (2) ***Efficient and Secure User Revocation***: On one hand, once a user is revoked from the group, the blocks signed by the revoked user can be efficiently re-signed. On the other hand, only existing users in the group can generate valid signatures on shared data, and the revoked user can no longer compute valid signatures on shared data. (3) ***Public Auditing***: The TPA can audit the integrity of shared data without retrieving the entire data from the cloud, even if some blocks in shared data have been re-signed by the cloud.

III. Preliminaries

In this section, we briefly introduce some cryptographic techniques we will use in this paper, including bilinear maps, homomorphic authenticators and proxy re-signatures.

A. Bilinear Maps

Let G_1 and G_2 be two multiplicative cyclic groups of prime order p, g be a generator of G_1. Bilinear map e is a map e: $G_1 \times G_1 \rightarrow G_2$ with the following properties: 1) **Computability**: there exists an efficient algorithm for computing map e. 2) **Bilinearity**: for all $u, v \in G_1$ and $a, b \in Z_p$, $e(u^a, v^b) = e(u, v)^{ab}$. 3) **Non-degeneracy**: $e(g, g) \neq 1$.

B. Complexity Assumptions

Definition 1: **Computational Diffie-Hellman (CDH) Problem.** For $a, b \in Z_p$, given $g, g^a, g^b \in G_1$ as input, output $g^{ab} \in G_1$.

The CDH assumption holds in G_1 if it is computationally infeasible to solve the CDH problem in G_1.

Definition 2: **Discrete Logarithm (DL) Problem.** For $a \in Z_p$, given $g, g^a \in G_1$ as input, output a.

The DL assumption holds in G_1 if it is computationally infeasible to solve the DL problem in G_1

C. Homomorphic Authenticators

Homomorphic authenticators [2], also called homomorphic verifiable tags, allow a public verifier to check the integrity of data stored in the cloud without downloading the entire data. They have been widely used in the previous public auditing mechanisms [2]–[9]. Besides *unforgeability* (only a user with a private key can generate valid signatures), a *homomorphic authenticable signature* scheme, which denotes a homomorphic authenticator scheme based on signatures, should also satisfy the following properties:

Let (pk, sk) denote the signer's public/private key pair, σ_1 denote the signature on block $m_1 \in Z_p$, and σ_2 denote the signature on block $m_2 \in Z_p$.

- ***Blockless verifiability:*** Given σ_1 and σ_2, two random values α_1, α_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 m_1 and m_2.
- ***Non-malleability:*** Given m_1 and m_2, σ_1 and σ_2, two random values α_1, α_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 combining σ_1 and σ_2.

Blockless verifiability enables a verifier to audit the correctness of data in the cloud with only a linear combination of all the blocks, while the entire data does not need to be downloaded to the verifier. Non-malleability indicates that an untrusted party cannot generate valid signatures on combined blocks by combining existing signatures.

D. Proxy Re-signatures

Proxy re-signatures, first proposed by Blaze *et al*. [11], allow a *semi-trusted* proxy to act as a translator of signatures between two users, for example, Alice and Bob. More specifically, the proxy is able to convert a signature of Alice into a signature of Bob on the same block. Meanwhile, the proxy is not able to learn any private keys of the two users, which means it cannot sign any block on behalf of either Alice or Bob. In this paper, to improve the efficiency of user revocation, we propose to let the cloud to act as the proxy and convert signatures for users.

IV. Homomorphic Authenticable Proxy Re-signatures

In this section, we first present a new proxy re-signature scheme, which satisfies the property of blockless verifiability and non-malleability. Then, we will describe how to construct our public auditing mechanism for shared data based on this proxy re-signature scheme in the next section.

A. HAPS: Construction

Because traditional proxy re-signature schemes [11], [12] are not blockless verifiable, if we directly apply these proxy re-signature schemes in the public auditing mechanism, then a verifier has to download the entire data to check data integrity, which will significantly reduce the efficiency of auditing. Therefore, we first propose a homomorphic authenticable proxy re-signature (HAPS) scheme, which is able to satisfy blockless verifiability and non-malleability. Our proxy re-signature scheme includes five algorithms: **KeyGen**, **ReKey**, **Sign**, **ReSign** and **Verify**.

Scheme Details: Let G_1 and G_2 be two groups of order p, g be a generator of G_1, $e : G_1 \times G_1 \rightarrow G_2$ be a bilinear map, w be a random element of G_1. The global parameters

are $(e, p, G_1, G_2, g, w, H)$, where H is a hash function with $H : \{0,1\}^* \rightarrow G_1$.

KeyGen. Given global parameters $(e, p, G_1, G_2, g, w, H)$, a user u_A selects a random $a \in Z_p$, and outputs his/her public key $pk_A = g^a$ and private key $sk_A = a$.

ReKey. The proxy generates a re-signing key $rk_{A\rightarrow B}$ as follows: (1) the proxy generates a random $r \in Z_p$ and sends it to user u_A; (2) user u_A computes and sends r/a to user u_B, where $sk_A = a$; (3) user u_B calculates and sends rb/a to the proxy, where $sk_B = b$; (4) the proxy recovers $rk_{A\rightarrow B} = b/a \in Z_p$. (We assume that private and authenticated channels exist between each pair of entities, and there is no collusion.)

Sign. Given private key $sk_A = a$, block $m \in Z_p$ and block identifier id, user u_A outputs the signature on block m as:

$$\sigma = (H(id)w^m)^a \in G_1. \quad (1)$$

ReSign. Given re-signing key $rk_{A\rightarrow B}$, public key pk_A, signature σ, block $m \in Z_p$ and block identifier id, the proxy checks that **Verify**$(pk_A, m, id, \sigma) \stackrel{?}{=} 1$. If the verification result is 0, the proxy outputs $\perp$; otherwise, it outputs

$$\sigma' = \sigma^{rk_{A\rightarrow B}} = (H(id)w^m)^{a\cdot b/a} = (H(id)w^m)^b \in G_1. \quad (2)$$

Verify. Given public key pk_A, block m, block identifier id, and signature σ, a verifier outputs 1 if

$$e(\sigma, g) = e(H(id)w^m, pk_A), \quad (3)$$

and 0 otherwise.

B. HAPS: Security Analysis

We now prove the correctness of the above proxy re-signature scheme. Based on the properties of bilinear maps, we have $e(\sigma, g) = e((H(id)w^m)^a, g) = e(H(id)w^m, pk_A)$. Then, we wish to show that our proxy re-signature scheme is unforgeable and homomorphic authenticable.

***Theorem** 1: It is computational infeasible to generate a forgery of a signature under HAPS.*

Proof: Following the standard security model defined in the previous proxy re-signature scheme [12], we show that our proxy re-signature scheme is able to resist forgery. The security of HAPS includes two aspects: *external security* and *internal security*. External security means an external adversary cannot generate a forgery of a signature; internal security means that the proxy cannot use its re-signature keys to sign on behalf of honest users. The logic of this proof is that if an external or internal adversary is able to generate a forgery of a signature under HAPS, then we could find an algorithm to solve the CDH problem, which however should be computational infeasible to solve under the CDH assumption.

External Security: An external adversary cannot generate a forgery of a signature. We show that if a (t', ϵ')-algorithm $\mathcal{A}$, operated by an external adversary, can generate a forgery of a signature under HAPS after making at most q_H hash queries, at most q_S signing queries, at most q_R re-signing queries, and requesting q_K public keys, then there exists a (t, ϵ)-algorithm $\mathcal{B}$ that can solve the CDH problem in G_1 with $t \leq t' + q_H c_{G_1} + q_S c_{G_1} + 2q_R c_P$ and $\epsilon \geq \epsilon'/q_H q_K$, where one exponentiation on G_1 takes time c_{G_1} and one pairing operation takes time c_P. On input (g, g^a, g^b), the CDH algorithm $\mathcal{B}$ simulates a proxy re-signature external security game for algorithm $\mathcal{A}$ as described in [13]. Due to space limitations, we omit the details of the external security game in this paper, further details of this proof can be found in our technical report [13].

Internal Security: The proxy cannot use its re-signature keys to sign on behalf of honest users. We now prove that, if a (t', ϵ')-algorithm $\mathcal{A}$, operated by the proxy, can generate a forgery of a signature after making at most q_H hash queries and q_S signing queries, then there exists a (t, ϵ)-algorithm $\mathcal{B}$ that can solve the CDH problem in G_1 with $t \leq t' + q_H c_{G_1} + q_S c_{G_1}$ and $\epsilon \geq \epsilon'/q_H q_K$. On input (g, g^a, g^b), the CDH algorithm $\mathcal{B}$ simulates a proxy re-signature internal security game for algorithm $\mathcal{A}$ as illustrated in [13]. Due to space limitations, we omit the details of the internal security game in this paper, further details of this proof can be found in our technical report [13].

Because under the external or internal security game, if a forgery of a signature can be generated, then we can find an algorithm to solve the CDH problem in G_1, which contradicts to the assumption that the CDH problem is computational infeasible in G_1. Therefore, it is computational infeasible to generate a forgery of a signature under HAPS. ■

***Theorem** 2: HAPS is a homomorphic authenticable proxy re-signature scheme.*

Proof: As we introduced in Section III, to prove HAPS is homomorphic authenticable, we need to show HAPS is not only blockless verifiable but also non-malleable. In addition, we also need to prove that the re-signing of the proxy does not affect these two properties, which means the signatures re-signed by the proxy are also blockless verifiable and non-malleable.

Given user u_a's public key pk_A, two random numbers y_1, $y_2 \in Z_p$, two identifiers id_1 and id_2, and two signatures σ_1 and σ_2 signed by user u_a, a verifier is able to check the correctness of a block $m' = y_1 m_1 + y_2 m_2$ by verifying

$$e(\sigma_1^{y_1} \cdot \sigma_2^{y_2}, g) \stackrel{?}{=} e(H(id_1)^{y_1} H(id_2)^{y_2} w^{m'}, pk_A), \quad (4)$$

without knowing block m_1 and block m_2. Based on the properties of bilinear maps, the correctness of the above equation can be proved as:

$$\begin{aligned} e(\sigma_1^{y_1} \cdot \sigma_2^{y_2}, g) &= e(H(id_1)^{y_1} w^{y_1 m_1} H(id_2)^{y_2} w^{y_2 m_2}, g^a) \\ &= e(H(id_1)^{y_1} H(id_2)^{y_2} w^{m'}, pk_A). \end{aligned}$$

It is clear that HAPS can support blockless verifiability.

Meanwhile, an adversary, who does not have private key $sk_A = a$, cannot generate a valid signature σ' for a combined block $m' = y_1 m_1 + y_2 m_2$ by combining σ_1 and σ_2 with y_1 and y_2. The hardness of this problem lies in the fact that H must be a one-way hash function (given every input, it is easy to compute; however, given the image of a random input, it is hard to invert).

More specifically, if we assume this adversary can generate a valid signature σ' for the combined block m' by combining σ_1 and σ_2, we have

$$\begin{cases} \sigma' = \sigma_1^{y_1} \cdot \sigma_2^{y_2} \\ \sigma_1^{y_1} \cdot \sigma_2^{y_2} = (H(id_1)^{y_1} H(id_2)^{y_2} w^{m'})^a \\ \sigma' = (H(id')w^{m'})^a \end{cases}$$

and we can further learn that $H(id') = H(id_1)^{y_1} H(id_2)^{y_2}$. Then, given a value of $h = H(id_1)^{y_1} H(id_2)^{y_2}$, we can easily find a block identifier id' so that $H(id') = h$, which contradicts to the assumption that H is a one-way hash function.

Because the construction and verification of the signatures re-signed by the proxy are as the same as the signatures computed by users, we can also prove that the signatures re-signed by the proxy are blockless verifiable and non-malleable in the same way illustrated above. Therefore, HAPS is a homomorphic authenticable proxy re-signature scheme. ■

V. Public Auditing with Efficient User Revocation

A. Overview

Based on the new proxy re-signature scheme and its properties illustrated in the previous section, we now present our public auditing mechanism for shared data with efficient user revocation. In our mechanism, we allow the cloud to perform as the proxy and translate signatures for users in the group. The original user acts as the group manager, who is able to revoke misbehaving users from the group.

B. Support Dynamic Data

To build the entire mechanism, another issue we need to consider is how to support dynamic data during public auditing. Because the computation of a signature includes the block identifier, conventional methods — which use the index of a block as the block identifier — are not efficient for supporting dynamic data [7], [8]. Specifically, if a single block is inserted or deleted, the indices of blocks that after this modified block are all changed, and the change of those indices requires the user to re-compute signatures on those blocks, even though the content of those blocks are not changed. Further explanation and corresponding figures can be found in our technical report [13].

By leveraging index hash tables [7], [8], we allow a user to modify a single block efficiently without changing block identifiers of other blocks (as presented in Fig. 4 and Fig. 5). More specifically, a block identifier, which is unique in the index hash table, is described as $id_i = \{v_i||r_i||s_i\}$, where $v_i \in \mathbb{N}^*$ is the *virtual index* of this block, r_i is computed as $r_i = H'(m_i||v_i)$ with a collision-resistance hash function $H' : \{0,1\}^* \rightarrow Z_q$, and s_i is the *signer id* of block m_i. Different from our previous work [8], in this paper, each block identifier contains a signer id to distinguish the identity of the signer. It is because, in this proposed mechanism, the cloud needs to know the identity of the signer on each block, so that it can easily distinguish which block needs to be re-signed during user revocation. The virtual indices ensure that all the blocks are in the right order. For instance, if $v_i > v_j$, then block m_i is one of the blocks that are after block m_j in shared data. The initial virtual index of block m_i is computed as $v_i = i \cdot \delta$, where $\delta \in \mathbb{N}^*$ is a system parameter decided by the original user and $\delta \geq 2$. If a new block m_i' is inserted, the virtual index of block m_i' is computed as $v_i' = \lfloor (v_{i-1} + v_i)/2 \rfloor$. Clearly, if block m_i and m_{i+1} are both initially created by the original user, the maximal number of inserted blocks that is allowed between block m_i and m_{i+1} is $\delta - 1$. The original user should choose a proper value of δ based on the related information of shared data, such as the number of users in the group, the type of the data file, the topic of the content in shared data, etc, so that users can perform insert operations on shared data properly.

No.	Block	V	R	S
1	m_1	δ	r_1	s_1
2	m_2	2δ	r_2	s_2
3	m_3	3δ	r_3	s_3
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
n	m_n	$n\delta$	r_n	s_n

Insert →

No.	Block	V	R	S
1	m_1	δ	r_1	s_1
2	m_2'	$\lfloor 3\delta/2 \rfloor$	r_2'	s_2'
3	m_2	2δ	r_2	s_2
4	m_3	3δ	r_3	s_3
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$n+1$	m_n	$n\delta$	r_n	s_n

Fig. 4. Insert block m_2' into shared data using an index hash table as block identifiers.

No.	Block	V	R	S
1	m_1	δ	r_1	s_1
2	m_2	2δ	r_2	s_2
3	m_3	3δ	r_3	s_3
4	m_4	4δ	r_4	s_4
5	m_5	5δ	r_5	s_5
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
n	m_n	$n\delta$	r_n	s_n

Update →

Delete →

No.	Block	V	R	S
1	m_1'	δ	r_1'	s_1'
2	m_2	2δ	r_2	s_2
3	m_4	4δ	r_4	s_4
4	m_5	5δ	r_5	s_5
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$n-1$	m_n	$n\delta$	r_n	s_n

Fig. 5. Update block m_1 and delete block m_3 in shared data using an index hash table as block identifiers.

C. Construction of Our Public Auditing Mechanism

Our public auditing mechanism includes six algorithms: **KeyGen**, **ReKey**, **Sign**, **ReSign**, **ProofGen**, **ProofVerify**. In **KeyGen**, every user in the group generates his/her public key and private key. In **ReKey**, the cloud computes a re-signing key for each pair of users in the group. When the original user creates shared data in the cloud, he/she computes a signature on each block as in **Sign**. After that, if a user in the group modifies a block in shared data, the signature on the modified block is also computed as in **Sign**. In **ReSign**, a user is revoked from the group, and the cloud re-signs the blocks, which were previously signed by this revoked user, with a re-signing key. The cloud is able to generate a proof of possession of shared data in **ProofGen**. In **ProofVerify**, a public verifier is able to check the correctness of a proof.

In **ReSign**, without loss of generality, we assume that the cloud always converts signatures of a revoked user into signatures of the original user. The reason is that the original user acts as the group manager, and we assume he/she is secure

in our mechanism. Another way to decide which re-signing key should be used when a user is revoked from the group, is to create a priority list (PL). Every existing user's id is in the PL and listed in the order of priority. When the cloud needs to decide which existing user the signatures should be converted into, the first user shown in the PL is selected. To ensure the correctness of the PL, it should be signed with the original user's private key.

Scheme Details: Let G_1 and G_2 be two groups of order p, g be a generator of G_1, $e : G_1 \times G_1 \to G_2$ be a bilinear map, w be a random element of G_1. The global parameters are $(e, p, G_1, G_2, g, w, H, H')$, where H is a hash function with $H : \{0,1\}^* \to G_1$ and H' is a hash function with $H' : \{0,1\}^* \to Z_q$. The total number of blocks in shared data is n, and shared data is described as $M = (m_1, ..., m_n)$. The total number of users in the group is d.

KeyGen. For user u_i, he/she generates a random $x_i \in Z_p$, and outputs his/her public key $pk_i = g^{x_i}$ and private key $sk_i = x_i$. Without loss of generality, we assume user u_1 is the original user, who is the creator of shared data. The original user also creates a user list (UL), which contains ids of all the users in the group. The user list is public and signed by the original user.

ReKey. The cloud generates a re-signing key $rk_{i\to j}$ as follows: (1) the cloud generates a random $r \in Z_p$ and sends it to user u_i; (2) user u_i sends r/x_i to user u_j, where $sk_i = x_i$; (3) user u_j sends rx_j/x_i to the cloud, where $sk_j = x_j$; (4) the cloud recovers $rk_{i\to j} = x_j/x_i \in Z_p$. (We still assume that private and authenticated channels exist between each pair of entities, and there is no collusion.)

Sign. Given private key $sk_i = x_i$, block $m_k \in Z_p$ in shared data M and its block identifier id_k, where $k \in [1, n]$, user u_i outputs the signature on block m_k as:

$$\sigma_k = (H(id_k)w^{m_k})^{x_i} \in G_1.$$

ReSign. When user u_i is revoked from the group, the cloud is able to convert signatures of user u_i into signatures of user u_j on the same block. More specifically, given re-signing key $rk_{i\to j}$, public key pk_i, signature σ_k, block m_k and block identifier id_k, the cloud first checks that $e(\sigma_k, g) \stackrel{?}{=} e(H(id_k)w^{m_k}, pk_i)$. If the verification result is 0, the cloud outputs $\perp$; otherwise, it outputs

$$\sigma'_k = \sigma_k^{rk_{i\to j}} = (H(id_k)w^{m_k})^{x_i \cdot x_j/x_i} = (H(id_k)w^{m_k})^{x_j}.$$

After the re-signing, the original user removes user u_i's id from UL and signs the new UL.

ProofGen. To audit the integrity of shared data, the TPA generates an auditing message as follows:

1) Randomly picks a c-element subset L of set $[1, n]$ to locate the c selected random blocks that will be checked in this auditing task.
2) Generates a random $y_l \in Z_q$, for $l \in L$ and q is a much smaller prime than p.
3) Outputs an auditing message $\{(l, y_l)\}_{l\in L}$, and sends it to the cloud.

After receiving an auditing message, the cloud generates a proof of possession of shared data M. More concretely,

1) The cloud divides set L into d subset $L_1, ..., L_d$, where L_i is the subset of selected blocks signed by user u_i. And the number of elements in subset L_i is c_i. Clearly, we have $c = \sum_{i=1}^{d} c_i$, $L = L_1 \cup ... \cup L_d$ and $L_i \cap L_j = \varnothing$, for $i \neq j$.
2) For each set L_i, the cloud computes $\alpha_i = \sum_{l\in L_i} y_l m_l \in Z_p$ and $\beta_i = \prod_{l\in L_i} \sigma_l^{y_l} \in G_1$.
3) The cloud outputs an auditing proof $\{\boldsymbol{\alpha}, \boldsymbol{\beta}, \{id_l\}_{l\in L}\}$, and sends it to the verifier, where $\boldsymbol{\alpha} = (\alpha_1, ..., \alpha_d)$ and $\boldsymbol{\beta} = (\beta_1, ..., \beta_d)$.

ProofVerify. With an auditing proof $\{\boldsymbol{\alpha}, \boldsymbol{\beta}, \{id_l\}_{l\in L}\}$, an auditing message $\{(l, y_l)\}_{l\in L}$, and all the existing users' public keys $(pk_1, ..., pk_d)$, the TPA checks the correctness of this auditing proof as

$$e(\prod_{i=1}^{d} \beta_i, g) \stackrel{?}{=} \prod_{i=1}^{d} e(\prod_{l\in L_i} H(id_l)^{y_l} \cdot w^{\alpha_i}, pk_i). \quad (5)$$

If the result is 1, the verifier believes that the integrity of all the blocks in shared data M is correct. Otherwise, the verifier outputs 0.

Discussion: As presented in our mechanism above, the TPA selects a number of random blocks instead of choosing all the blocks in shared data, which can improve the efficiency of auditing. Previous work [2] has already proved that a verifier is able to detect the polluted blocks with a high probability by selecting a small number of random blocks. More specifically, when shared data contains $n = 1,000,000$ blocks, if 1% of all the blocks are corrupted, a verifier can detect these polluted blocks with a probability greater than 99% or 95%, where the number of selected blocks c is 460 or 300, respectively.

D. Scalability of Our Public Auditing Mechanism

In many cases, the TPA may receive amount of auditing requests from different users in a very short time period. Clearly, asking the TPA to perform these auditing requests one by one is not efficient. Therefore, to improve the scalability of our public auditing mechanism in such cases, we can further extend our proposed public auditing mechanism to support batch auditing [6] by utilizing the properties of bilinear maps. With batch auditing, the TPA can perform multiple auditing tasks simultaneously. Due to space limitations, further discussion and explanation about batch auditing of our mechanism can be found in our technical report [13]. We also discuss how to improve the scalability of our mechanism by reducing the total number of re-signing keys that the cloud needs to manage when cloud data is shared by a very large number of group users in the full version of this paper [13].

E. Security Analysis of Our Public Auditing Mechanism

***Theorem** 3: Given shared data M and its signatures, a verifier is able to correctly check the integrity of shared data M.*

Proof: To prove the correctness of our mechanism is equivalent of proving Equation (5) is correct. Based on the

properties of bilinear maps, the correctness of Equation (5) is presented as the following:

$$\begin{aligned} e(\prod_{i=1}^{d}\beta_i, g) &= \prod_{i=1}^{d} e(\prod_{l\in L_i}\sigma_l^{y_l}, g) \\ &= \prod_{i=1}^{d} e(\prod_{l\in L_i}(H(id_l)w^{m_l})^{x_i y_l}, g) \\ &= \prod_{i=1}^{d} e(\prod_{l\in L_i}H(id_l)^{y_l}\cdot\prod_{l\in L_i}w^{m_l y_l}, g^{x_i}) \\ &= \prod_{i=1}^{d} e(\prod_{l\in L_i}H(id_l)^{y_l}\cdot w^{\alpha_i}, pk_i). \end{aligned}$$

■

***Theorem** 4: For the cloud, it is computational infeasible to generate a forgery of an auditing proof under our mechanism.*

Proof: Following the security model and security game defined in [3], [8], we can prove that, if the cloud could win a security game, named Game 1, by forging an auditing proof on incorrect shared data, then we can find a solution to the Discrete Logarithm problem in G_1 with a probability of $1 - 1/p$, which contradicts to the DL assumption. We define Game 1 as follows:

Game 1: The TPA sends an auditing message $\{(l, y_l)\}_{l\in L}$ to the cloud, the auditing proof on correct shared data M should be $\{\boldsymbol{\alpha}, \boldsymbol{\beta}, \{id_l\}_{l\in L}\}$, which should pass the verification with Equation (5). However, the cloud generates a proof on incorrect shared data M' as $\{\boldsymbol{\alpha}', \boldsymbol{\beta}, \{id_l\}_{l\in L}\}$, where $\boldsymbol{\alpha}' = (\alpha_1, ..., \alpha_d)$, $\alpha_i' = \sum_{l\in L_i} y_l m_l'$, for $i \in [1, d]$, and $M \neq M'$. Define $\Delta\alpha_i = \alpha_i' - \alpha_i$ for $1 \leq i \leq d$, and at least one element of $\{\Delta\alpha_i\}_{1\leq i\leq d}$ is nonzero. If this proof still pass the verification performed by the TPA, then the cloud wins this game. Otherwise, it fails.

We first assume that the cloud wins the game. Then, according to Equation (5), we have

$$e(\prod_{i=1}^{d}\beta_i, g) = \prod_{i=1}^{d} e(\prod_{l\in L_i}H(id_l)^{y_l}\cdot w^{\alpha_i'}, pk_i).$$

Because $\{\boldsymbol{\alpha}, \boldsymbol{\beta}, \{id_l\}_{l\in L}\}$ is a correct auditing proof, we have

$$e(\prod_{i=1}^{d}\beta_i, g) = \prod_{i=1}^{d} e(\prod_{l\in L_i}H(id_l)^{y_l}\cdot w^{\alpha_i}, pk_i).$$

Based on the properties of bilinear maps, we can learn that

$$\prod_{i=1}^{d} w^{\alpha_i x_i} = \prod_{i=1}^{d} w^{\alpha_i' x_i}, \quad \prod_{i=1}^{d} w^{x_i \Delta\alpha_i} = 1.$$

Because G_1 is a cyclic group, then for two elements $u, v \in G_1$, there exists $x \in Z_p$ that $v = u^x$. Without loss of generality, given u, v, each w^{x_i} can generated as $w^{x_i} = u^{\xi_i} v^{\gamma_i} \in G_1$, where ξ_i and γ_i are random values of Z_p. Then, we have

$$1 = \prod_{i=1}^{d}(u^{\xi_i} v^{\gamma_i})^{\Delta\alpha_i} = u^{\sum_{i=1}^{d}\xi_i\Delta\alpha_i} \cdot v^{\sum_{i=1}^{d}\gamma_i\Delta\alpha_i}.$$

Clearly, we can find a solution to the Discrete Logarithm problem. Given $u, v = u^x \in G_1$, we can output

$$v = u^{-\frac{\sum_{i=1}^{d}\xi_i\Delta\alpha_i}{\sum_{i=1}^{d}\gamma_i\Delta\alpha_i}}, \quad x = -\frac{\sum_{i=1}^{d}\xi_i\Delta\alpha_i}{\sum_{i=1}^{d}\gamma_i\Delta\alpha_i},$$

unless the denominator is zero. However, as we defined in Game 1, at least one of element in $\{\Delta\alpha_i\}_{1\leq i\leq d}$ is nonzero, and γ_i is a random element of Z_p, therefore, the denominator is zero with a probability of $1/p$, which is negligible because p is a large prime. Then, we can find a solution to the Discrete Logarithm problem with a probability of $1 - 1/p$, which contradicts to the assumption that Discrete Logarithm problem is computationally infeasible in G_1. ■

F. Efficient and Secure User Revocation

We argue that our mechanism is efficient and secure during user revocation. It is efficient because when a user is revoked from the group, the cloud can re-sign blocks that were previously signed by the revoked user with a re-signing key, while an existing user does not have to download those blocks, recompute signatures on those blocks and upload new signatures to the cloud. The re-signing preformed by the cloud improves the efficiency of user revocation and saves communication and computation resources for existing users.

The user revocation is secure because only existing users are able to sign the blocks in shared data. As analyzed in Theorem 1, even with a re-signing key, the cloud cannot generate a valid signature for an arbitrary block on behalf of an existing user. In addition, after being revoked from the group, a revoked user is no longer in the user list, and can no longer generate valid signatures on shared data.

VI. Performance

In this section, we first discuss the communication and computation cost of our mechanism. Then we evaluate the performance of our mechanism in experiments.

A. Communication Cost

According to the description in Section V, the size of an auditing message $\{(l, y_l)\}_{l\in L}$ is $c \cdot (|n| + |q|)$ bits, where c is the number of selected blocks, $|n|$ is the size of an element of set $[1, n]$ and $|q|$ is the size of an element of Z_q. The size of an auditing proof $\{\boldsymbol{\alpha}, \boldsymbol{\beta}, \{id_l\}_{l\in L}\}$ is $2d \cdot |p| + c \cdot (|id|)$ bits, where d is the number of existing users in the group, $|p|$ is the size of an element of G_1 or Z_p, $|id|$ is the size of a block identifier. Therefore, the total communication cost of an auditing task is $2d \cdot |p| + c \cdot (|id| + |n| + |q|)$ bits.

B. Computation Cost

As shown in **ReSign** of our mechanism, the cloud first verifies the correctness of the original signature on a block, and then computes a new signature on the same block with a re-signing key. The computation cost of re-signing a block in the cloud is $2\texttt{Exp}_{G_1} + \texttt{Mul}_{G_1} + 2\texttt{Pair} + \texttt{Hash}_{G_1}$, where $\texttt{Exp}_{G_1}$ denotes one exponentiation in G_1, $\texttt{Mul}_{G_1}$ denotes one multiplication in G_1, $\texttt{Pair}$ denotes one pairing operation on

$e : G_1 \times G_1 \rightarrow G_2$, and $\texttt{Hash}_{G_1}$ denotes one hashing operation in G_1. The cloud can further reduce the computation cost of the re-signing on a block to $\texttt{Exp}_{G_1}$ by directly re-signing it without verification. The public auditing performed by the TPA ensures that the re-signed blocks are correct. Based on Equation (5), the computation cost of an auditing task in our mechanism is $(c+d)\texttt{Exp}_{G_1} + (c+2d)\texttt{Mul}_{G_1} + (d+1)\texttt{Pair} + d\texttt{Mul}_{G_2} + c\texttt{Hash}_{G_1}$.

C. Experimental Results

In this section, we evaluate the performance of our mechanism in experiments. We utilize Pairing Based Cryptography Library (PBC)[1] to implement cryptographic operations in our mechanism. All the experiments are tested under Ubuntu with an Intel Core i5 2.5 GHz Processor and 4 GB Memory over $1,000$ times. In the following experiments, we assume the size of an element of G_1 or Z_p is $|p| = 160$ bits, the size of an element of Z_q is $|q| = 80$ bits, the size of a block identifier is $|id| = 80$ bits, and the total number of blocks in shared data is $n = 1,000,000$. By utilizing aggregation methods from [3], [8], the size of each block can be set as 2 KB, then the total size of shared data is 2 GB.

1) Performance of User Revocation: As introduced in Section I, the main purpose of our mechanism is to improve the efficiency of user revocation. Without our mechanism, to revoke a user in the group, an existing user needs to download the blocks were previously signed by the revoked user, verify the correctness of these blocks, re-compute signatures on these blocks and upload the new signatures. In this experiment, we assume the download speed and upload speed for the data storage and sharing services is 1Mbps and 500Kbps, respectively. We also assume the cloud and an existing user leverage the same type of machine (Intel Core i5 2.5 GHz Processor and 4 GB Memory) to perform user revocation. Let k denote the number of re-signed blocks during user revocation.

The performance of our mechanism during user revocation is presented in Fig. 6. The cloud is able to not only efficiently re-sign blocks but also save existing users' computation and communication resources. As shown in Fig. 6, when the number of re-signed blocks is 500, which is only 0.05% of the total number of blocks, the cloud in our mechanism can re-sign these blocks within 15 seconds. In contrast, without our mechanism, an existing user needs about 22 seconds to re-sign the same number of blocks by herself. Besides, the 500 re-signed blocks that this existing user downloaded costs her extra bandwidth during user revocation. Both of the two revocation time are linearly increasing with an increase of k—the number of re-signed blocks. Since we assume the cloud and an existing user have the same level of computation resource in this experiment, it is easy to see that the gap in terms of revocation time between the two lines in Fig. 6 is mainly introduced by downloading the re-signed blocks. In a practical cloud environment, the cloud should have more powerful computation capabilities than personal devices, which allows the cloud to finish the re-signing on data even sooner.

[1]http://crypto.stanford.edu/pbc/

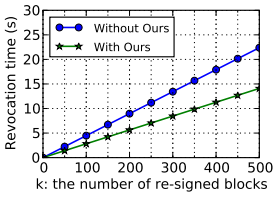

Fig. 6. Impact of k on revocation time (s).

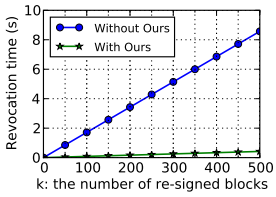

Fig. 7. Impact of k on revocation time without verification (s).

In addition, as we analyzed before, the cloud can even directly re-sign data without verification, which can further improve the efficiency of re-signing about 100 times. More specifically, the re-signing time on one block with verification is 28.19 milliseconds while the one without verification is only 0.28 milliseconds. Note that due to the existence of transmission errors in networks, it is not a good idea to allow an existing user to re-sign the blocks without verifying them. Even if an existing user directly re-signs the blocks without verification, compared to our mechanism, this user still needs to spend some extra time to download the blocks. As illustrated in Fig. 7, when the number of re-signed blocks is still 500, the cloud in our mechanism can re-sign these blocks in about 0.14 seconds; while an existing user needs about 8.43 seconds by herself. With the comparison between Fig. 6 and Fig. 7, we can see that the verification on original signatures before re-signing is one of the main factors that can slow down the entire user revocation process. Meanwhile, as shown in Fig. 6 and Fig. 7, the key advantage of our mechanism is that we can improve the efficiency of user revocation and release existing users from the communication and computation burden introduced by user revocation.

2) Performance of Auditing: We can see from Fig. 8 and Fig. 9 that, in order to maintain a higher detection probability, a verifier needs more time and communication overhead to finish the auditing task on shared data. Meanwhile, the auditing time (the time that the TPA needs to verify the correctness of an auditing proof based on Equation (5)) is linearly increasing with the number of existing users in the group. Our mechanism allows a verifier to efficiently audit the correctness of shared data without retrieving the entire data from the cloud. More specifically, when $c = 460$ and $d = 10$, the communication cost of an auditing task (the communication cost that the TPA requires during an auditing task) is about 11.9 KB, and the auditing time of the entire data is only about 300 milliseconds.

VII. 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

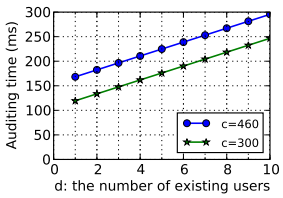

Fig. 8. Impact of d on auditing time (ms).

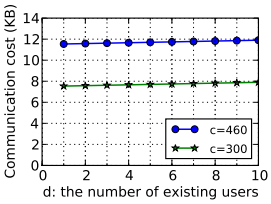

Fig. 9. Impact of d on communication cost (KB).

without retrieving the entire data, which is referred to as public verifiability or public auditing. Shacham and Waters [3] designed an improved PDP scheme based on BLS signatures. To support dynamic operations on data during auditing, Ateniese *et al.* [14] presented another PDP mechanism based on symmetric keys. However, it is not publicly verifiable and only provides a user with a limited number of verification requests. Wang *et al.* utilized the Merkle Hash Tree to support fully dynamic operations in a public auditing mechanism. Erway *et al.* [15] introduced Dynamic Provable Data Possession by using authenticated dictionaries, which are based on rank information. Zhu *et al.* [7] 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.

Wang *et al.* [4] leveraged homomorphic tokens to ensure the correctness of erasure code-based data distributed on multiple servers. To minimize the communication overhead in the phase of data repair, Chen *et al.* [16] introduced a mechanism for auditing the correctness of data with the multi-server scenario, where these data are encoded with network coding. More recently, Cao *et al.* [9] constructed an LT code-based secure cloud storage mechanism. Compared to previous mechanisms [4], [16], this mechanism can avoid high decoding computation costs for data users and save computation resources for online data owners during data repair.

When a third party auditor (TPA) is introduced into a public auditing mechanism in the cloud, both the content of data and the identities of signers are private information to users, and should be preserved from the TPA. The public mechanism proposed by Wang *et al.* [6] 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 also extended their mechanism to support batch auditing. Our recent work [8] first proposed a mechanism for public auditing shared data in the cloud for a group of users. With ring signature-based homomorphic authenticators, the TPA can verify the integrity of shared data but is not able to reveal the identity of the signer on each block. The auditing mechanism in [10] is designed to preserve identity privacy for a large number of users. However, it fails to support public auditing.

VIII. Conclusions

In this paper, we proposed a new public auditing mechanism for shared data with efficient user revocation in the cloud. When a user in the group is revoked, we allow the cloud to re-sign blocks that were signed by the revoked user with proxy re-signatures. Experimental results show that the cloud can improve the efficiency of user revocation, and existing users in the group can save a significant amount of computation and communication resources during user revocation.

Acknowledgement

We are grateful to the anonymous reviewers for their helpful suggestions. This work is supported by the National Natural Science Foundation of China (No. 61272457 and 61003300), 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 (PCSIRT 1078).

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 *the Proceedings of ACM CCS 2007*, 2007, pp. 598–610.

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

[4] C. Wang, Q. Wang, K. Ren, and W. Lou, "Ensuring Data Storage Security in Cloud Computing," in *the Proceedings of ACM/IEEE IWQoS 2009*, 2009, pp. 1–9.

[5] Q. Wang, C. Wang, J. Li, K. Ren, and W. Lou, "Enabling Public Verifiability and Data Dynamic for Storage Security in Cloud Computing," in *the Proceedings of ESORICS 2009*. Springer-Verlag, 2009, pp. 355–370.

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

[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 *the Proceedings of ACM SAC 2011*, 2011, pp. 1550–1557.

[8] B. Wang, B. Li, and H. Li, "Oruta: Privacy-Preserving Public Auditing for Shared Data in the Cloud," in *the Proceedings of IEEE Cloud 2012*, 2012, pp. 295–302.

[9] N. Cao, S. Yu, Z. Yang, W. Lou, and Y. T. Hou, "LT Codes-based Secure and Reliable Cloud Storage Service," in *the Proceedings of IEEE INFOCOM 2012*, 2012, pp. 693–701.

[10] B. Wang, B. Li, and H. Li, "Knox: Privacy-Preserving Auditing for Shared Data with Large Groups in the Cloud," in *the Proceedings of ACNS 2012*, June 2012, pp. 507–525.

[11] M. Blaze, G. Bleumer, and M. Strauss, "Divertible Protocols and Atomic Proxy Cryptography," in *the Proceedings of EUROCRYPT 98*. Springer-Verlag, 1998, pp. 127–144.

[12] G. Ateniese and S. Hohenberger, "Proxy Re-signatures: New Definitions, Algorithms and Applications," in *the Proceedings of ACM CCS 2005*, 2005, pp. 310–319.

[13] B. Wang, B. Li, and H. Li, "Public Auditing for Shared Data with Efficient User Revoation in the Cloud," Xidian University, Xi'an, China, Tech. Rep., 2012. [Online]. Available: http://ste.xidian.edu.cn/lihui/cloud12.pdf

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

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

[16] B. Chen, R. Curtmola, G. Ateniese, and R. Burns, "Remote Data Checking for Network Coding-based Distributed Stroage Systems," in *the Proceedings of ACM CCSW 2010*, 2010, pp. 31–42.