Providing Efficient Methods for Auditing Shared Data in the Cloud

Neha Sharon A, Burugapalli Roja
Department of Computer Science and Engineering
T John Institute of Technology
Bangalore,India

Bhavya N Javagal
Asst Prof,Dept of CSE
T.John Institute Of Technology
Bangalore,India

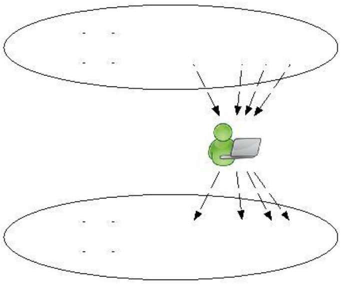
***Abstract* - With data storage and sharing services in the cloud, users can easily modify and share data as a group. To ensure shared data integrity can be verified publicly, users in the group need to compute signatures on all the blocks in shared data. Different blocks in shared data are generally 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. This paper proposes a novel public auditing mechanism for the integrity of shared data with efficient user revocation in mind. By utilizing the idea of proxy re-signatures, the cloud is allowed 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. Moreover, this mechanism is able to support batch auditing by verifying multiple auditing tasks simultaneously. Experimental results show that this mechanism can significantly improve the efficiency of user revocation.**

1. INTRODUCTION

WITH data storage and sharing services (such as Dropbox and 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 [2], [3]. To protect the integrity of data in the cloud, a number of mechanisms [3]–[15] 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 all the signatures. One of the most significant and common features of these mechanisms is to allow a public verifier to efficiently check data integrity in the cloud without downloading the entire data, referred to as public auditing (or denoted as Provable Data Pos-session [3]). This public verifier could be a client who would like to utilize cloud data for particular purposes (e.g., search, computation, data mining, etc.) or a third-party auditor (TPA) who is able to provide verification services on data integrity to users. Most of the previous works [3]–[13] focus on auditing the integrity of personal data. Different from these works, several recent works [14], [15] focus on how to preserve identity privacy from public verifiers when auditing the integrity of shared data. Unfortunately, none of the above mechanisms, 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, dif-ferent 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 [16]. 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. As a result, the integrity of the entire data can still be verified with the public keys of existing users only.

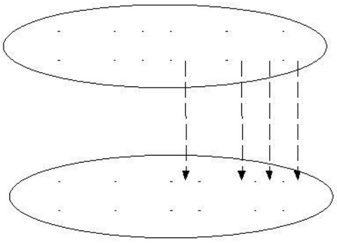
Since shared data is outsourced to the cloud and users no longer store it on local devices, a straightforward method to re-compute these signatures during user revo-cation (as shown in Fig. 1) is to ask an existing user (i.e., Alice) to first download the blocks previously signed by the revoked user (i.e., Bob), 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 re-sources 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 mem-bership of the group is frequently changing. To make this matter even worse, existing users may access their data sharing services provided by the cloud with re-source-limited devices, such as mobile phones, which further prevents existing users from maintaining the correctness of shared data efficiently during user revocation.



CLOUD	
	Before BOB is revoked
	1. Download blocks
A LICE	2. Verify blocks
	3. Re-compute signatures
	4. Upload signa-tures
CLOUD	
	After BOB is revoked

A A block signed by ALICE B A block signed by BOB

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 — audit-ing 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.Moreover, our proposed mechanism is scalable, which



CLOUD

A A A A A B A B B B	Before Bob is revoked
	Cloud re-signs blocks with
	a re-signing key
CLOUD	
A A A A A A A A A A	After Bob is revoked
A A block signed by Alice B	A block signed by Bob

Fig. 2. When Bob is revoked, the cloud re-signs the blocks that were previously signed by Bob with a re-signing key.indicates it is not only able to efficiently support a large number of users to share data and but also able to han-dle multiple auditing tasks simultaneously with batch auditing. In addition, by taking advantages of Shamir Secret Sharing [18], we can also extend our mechanism into the multi-proxy model to minimize the chance of the misuse on re-signing keys in the cloud and improve the reliability of the entire mechanism.

The remainder of this paper is organized as follows: In Section 2, we present the system model, security model and design goals. Then, we introduce several preliminar-ies in Section 3. Detailed design and security analysis of our mechanism are presented in Section 4 and Section 5. We discuss the extension of our mechanism in Section 6, and evaluate the performance of our mechanism in Section 7 and Section 8. Finally, we briefly discuss related work in Section 9, and conclude this paper in Section 10.

2.PROBLEM STATEMENT

In this section, we describe the system and security model, and illustrate the design objectives of our pro-posed mechanism.

2.1 System and Security Model

As illustrated in Fig. 3, the system model in this paper includes three entities: the cloud, the public verifier, and users (who share data as a group). The cloud offers data storage and sharing services to the group. The public verifier, such as a client who would like to utilize cloud data for particular purposes (e.g., search, computation, data mining, etc.) or a third-party auditor (TPA) who can provide verification services on data integrity, aims to check the integrity of shared data via a challenge-and-response protocol with the cloud. In the 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 divided into a number of blocks. A user in the group can modify a block in shared data by performing an insert, delete or update operation on the block.

In this paper, we assume the cloud itself is semi-trusted, which means it follows protocols and does not pollute data integrity actively as a malicious adversary, but it may lie to verifiers about the incorrectness of shared data in order to save the reputation of its data services and avoid losing money on its data services. In addition, we also assume there is no collusion between the cloud and any user during the design of our mechanism. Generally, the incorrectness of share data under the above semi-trusted model can be introduced by hardware/software failures or human errors happened in the cloud. Con-sidering these factors, users do not fully trust the cloud with the integrity of shared data.

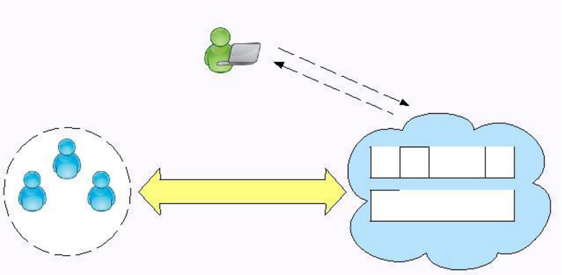


Fig. 3. The system model includes the cloud, the public verifier, and users.

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 should be re-signed by an existing user's private key, so that the correctness of the entire data can still be verified with the public keys of existing users only.

Alternative Approach. 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 [19], [20]. However, when a user is revoked, 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 decreases the efficiency of user revocation.

2.2 Design Objectives

Our proposed mechanism should achieve the follow-ing properties: (1) Correctness: The public verifier 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 public verifier 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. (4) **Scalability**: Cloud data can be efficiently shared among a large number of users, and the public verifier is able to handle a large number of auditing tasks simultaneously and efficiently.

3. PRELIMINARIES

In this section, we briefly introduce some prelimi-naries, including bilinear maps, security assumptions, homomorphic authenticators and proxy re-signatures.

3.1 Bilinear Maps

Let G1 and G2 be two multiplicative cyclic groups of prime order p, g be a generator of G1. Bilinear map e is a map e: G1 × G1 → G2 with the following properties: 1) **Computability**: there exists an efficient algorithm for computing map e. 2) **Bilinearity**: for all u, v ∈ G1 and a, b ∈ Zp, e(ua, vb) = e(u, v)ab. 3) Non-degeneracy: e(g, g) 6= 1.

3.2 Security Assumptions

The security of our mechanism is based on the follow-ing security assumptions.

Computational Diffie-Hellman (CDH) Problem. Let a, b ∈ Z∗ p, given g, ga, gb ∈ G1 as input, output gab ∈ G1.

Definition *1:* **Computational Diffie-Hellman (CDH) Assumption.** For any probabilistic polynomial time adversary ACDH, the advantage of adversary ACDH on solving the CDH problem in G1 is negligible, which is defined as

$$P\ r[ACDH(g, g^a, g^b) = (g^{ab}) : a, b \xleftarrow{R} Zp^*] \le ǫ.$$

For the ease of understanding, we can also say computing the CDH problem in G1 is computationally infeasible or hard under the CDH assumption.

Discrete Logarithm (DL) Problem. Let a ∈ Z∗ p, given g, ga ∈ G1 as input, output a.

Definition *2:* **Discrete Logarithm (DL) Assumption.** For any *probabilistic polynomial time* adversary ADL, the advantage of adversary ADL on solving the DL problem in G1 is negligible, which is defined as

$$P\ r[ADL(g, g^a) = (a) : a \xleftarrow{R} Zp^*] \le ǫ.$$

Similarly, we can also say computing the DL problem in G1 is computationally infeasible or hard under the DL assumption.

3.3 Homomorphic Authenticators

Homomorphic authenticators [3], also called homo-morphic verifiable tags, allow a public verifier to check the integrity of data stored in the cloud without down-loading the entire data. They have been widely used as building blocks in the previous public auditing mech-anisms [3]–[15], [19], [20]. Besides unforgeability (only a user with a private key can generate valid signatures), a homomorphic authenticable signature scheme, which de-notes a homomorphic authenticator scheme based on signatures, should also satisfy the following properties:

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

- **Blockless verifiability:** Given σ1 and σ2, two ran-dom values α1, α2 in Zp and a block m′ = α1m1 + α2m2 ∈ Zp, a verifier is able to check the correctness of block m′ without knowing m1 and m2.
- **Non-malleability:** Given m_1 and m_2, σ_1 and σ_2,

two	random values	α_1, α_2 in $Z_\mathbf{p}$ and a block
m′	$= \alpha_1 m_1 + \alpha_2 m_2$	$\in Z_\mathbf{p}$, a user, who does not

have private key sk, is not able to generate a valid signature σ′ on block m′ by combining σ1 and σ2.

Blockless verifiability enables a verifier to audit the correctness of data in the cloud with only a linear com-bination of all the blocks via a challenge-and-response protocol, while the entire data does not need to be down-loaded to the verifier. Non-malleability indicates that other parties, who do not possess proper private keys, cannot generate valid signatures on combined blocks by combining existing signatures.

3.4 Shamir Secret Sharing

An (s, t)-Shamir Secret Sharing scheme [18] (s ≥ 2t − 1), first proposed by Shamir, is able to divide a secret π into s pieces in such a way that this secret π can be easily recovered from any t pieces, while the knowledge of any t − 1 pieces reveals absolutely no information about this secret π.

The essential idea of an (s, t)-Shamir Secret Sharing scheme is that, a number of t points uniquely defines a t − 1 degree polynomial. Suppose we have the following t − 1 degree polynomial

$$f(x) = a_{\mathbf{t}-1}x^{\mathbf{t}-1} + \cdot \cdot \cdot + a_1x + a_0,$$

where at−1, ..., a1 $\overset{\mathbf{R}}{\leftarrow} \in Z_p^{*}$. Then, the secret is π = a0, and each piece of this secret is actually a point of polynomial f (x), i.e. (xi, f (xi)), for 1 ≤ i ≤ s. The secret π can be recovered by any t points of this t−1 degree polynomial f (x) with Lagrange polynomial interpolation. Shamir Se-cret Sharing is widely used in key management schemes [18] and secure multi-party computation [21].

$t \leq t' + q_\mathbf{H}\, c_{G_1} + q_\mathbf{S}\, c_{G_1} + 2q_\mathbf{R} c_\mathbf{P}$, $ϙ \geq ϙ'/q_\mathbf{H}\, q_\mathbf{K}$,

4 A NEW PROXY RE-SIGNATURE SCHEME

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

4.1 Construction of HAPS

Because traditional proxy re-signature schemes [17], [22] are not blockless verifiable, if we directly apply these proxy re-signature schemes in the public auditing mech-anism, then a verifier has to download the entire data to check the 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 algo-rithms: KeyGen, ReKey, Sign, ReSign and Verify. De-tails of each algorithm are described in Fig. 4. Similar as the assumption in traditional proxy re-signature schemes [17], [22], we assume that private channels (e.g., SSL) exist between each pair of entities in Rekey, and there is no collusion between the proxy and any user. Based on the properties of bilinear maps, the correctness of the verification in Verify can be presented as

Public Keys: As A requests the creation of system users, B guesses which one A will attempt a forgery against. Without loss of generality, we assume the target public key as pkv and set it as pkv = ga. For all other

non-negligible, then we can find an algorithm to solve the CDH problem in G1 with a non-negligible probabil-ity, which contradicts to the assumption that the CDH problem is computationally infeasible in G1. Therefore, it is computationally infeasible to generate a forgery of a signature in HAPS under the CDH assumption.

$$e(\sigma, g) = e((H(id)w^{\mathbf{m}})^{\mathbf{a}}, g) = e(H(id)w^{\mathbf{m}}, pk_\mathbf{A}).$$

4.2 Security Analysis of HAPS

Theorem *1: It is computationally infeasible to generate a forgery of a signature in HAPS as long as the CDH assumption holds.*

Proof: Following the standard security model de-fined in the previous proxy re-signature scheme [22], the security of HAPS includes two aspects: *external security* and *internal security*. External security means an exter-nal 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.

External Security: We show that if a (t′, ϙ′)-algorithm A, operated by an external adversary, can generate a forgery of a signature under HAPS with the time of t′ and advantage of ϙ′ after making at most $q_\mathbf{H}$ hash queries, $q_\mathbf{S}$ signing queries, $q_\mathbf{R}$ re-signing queries, and requesting at most $q_\mathbf{K}$ public keys, then there exists a (t, ϙ)-algorithm B that can solve the CDH problem in G_1 with where one exponentiation on G_1 takes time c_{G1} and one pairing operation takes time $c_\mathbf{P}$. Specifically, on input (g, $g^\mathbf{a}$, $g^\mathbf{b}$), the CDH algorithm B simulates a proxy re-signature security game for algorithm A as follows:

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

Proof: As we introduced in Section 3, to prove HAPS is homomorphic authenticable, we need to show HAPS is not only blockless verifiable but also non-malleable. Moreover, we also need to prove that the re-signing performed by the proxy does not affect these two properties.

Blockless Verifiability. 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 $m' = y_1m_1 + y_2m_2$ by verifying 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:

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

It is clear that HAPS can support blockless verifiability. Non-malleability. Meanwhile, an adversary, who does not have private key skA = a, cannot generate a valid signature σ′ for a combined block m′ = y1m1 + y2m2 by combining σ1 and σ2 with y1 and y2. 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

	σ'	$= \sigma_1^{y_1} \cdot \sigma_2^{y_2}$				′	
	y_1	y_2	= (H(id′	1	2		
	σ_1	$\cdot\ \sigma_2$		$1)^y$	$H(id_2)^y$	w^m	$)^a$
σ'	$= (H(id')w^{m})^{a}$						

and we can further learn that $H(id') = H(id_1)^{y_1} H(id_2)^{y_2}$. Then, that means, 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.

5.OVERVIEW

Based on the new proxy re-signature scheme and its properties in the previous section, we now present 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 (i.e., block m_j is indexed with j) — are not efficient for supporting dynamic data [8], [14]. 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. Each block is attached with a signature, a block identifier and a signer identifier.

By leveraging index hash tables [8], [14], we allow a user to modify a single block efficiently without chang-ing block identifiers of other blocks. The details of index hash tables are explained in Appendix A. Besides a block identifier and a signature, each block is also attached with a signer identifier (as shown in Fig. 6). A verifier can use a signer identifier to distinguish which key is required during verification, and the cloud can utilize it to determine which re-signing key is needed during user revocation.

5.1Construction

It includes six algorithms: **KeyGen**, **ReKey**, **Sign**, **ReSign**, **ProofGen**, **ProofVerify**. Details of are presented in Fig. 5.

In **KeyGen**, every user in the group generates his/her public key and private key. In **ReKey**, the cloud com-putes a re-signing key for each pair of users in the group. As argued in previous section, we still assume that private channels exist between each pair of entities

Game 1: The public verifier sends an auditing mes-sage $\{(l, \eta_l)\}_{l\in L}$ to the cloud, the auditing proof on correct shared data M should be $\{\alpha, \beta, \{id_l\}_{l\in L}\}$, which should pass the verification with Equation (2). How-ever, the cloud generates a proof on incorrect shared verification performed by the public verifier, then the cloud wins this game. Otherwise, it fails.

We first assume that the cloud wins the game. Then, according to Equation (2), we have

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

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

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

Based on the properties of bilinear maps, we learn that 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^{π_i} can generated as $w^{\pi_i} = u^{\xi_i} v^{\gamma_i} \in G_1$, where ξ_i and γ_i are random values of Z_p.
Clearly, we can find a solution to the DL problem. Given u, $v = u^x \in G_1$, we can output unless the denominator is zero. However, as we defined in Game 1, at least one of element in $\{\alpha_i\}_{1\le i\le d}$ 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 DL problem with a non-negligible probability of 1 − 1/p, which contradicts to the DL assumption in G_1. |

5.2 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, re-compute 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.

5.3 Limitations and Future Work

Remind that in Section 2, we assume there is no collusion between the cloud and any user in the design of our mechanism as the same as the assumption in some traditional proxy re-signatures [17], [22]. The reason is that, in our current design, if a revoked user (e.g., Bob with private key $sk_{\mathbf{b}}$) is able to collude with the cloud, who possesses a re-signing key (e.g., $rk_{\mathbf{a}\rightarrow\mathbf{b}} = sk_{\mathbf{a}}/sk_{\mathbf{b}}$, then the cloud and Bob together are able to easily reveal the private key of an existing user (e.g., Alice's private key $sk_{\mathbf{a}}$).

To overcome the above limitation, some proxy re-signature schemes with collusion resistance in [22], which can generate a re-signing key with a revoked user's public key and an existing user's private key, would be a possible solution. Specifically, a resigning key is computed as $rk_{\mathbf{a}\rightarrow\mathbf{b}} = pk^{SK}{}_{\mathbf{b}}{}^{a}$ by Alice, then even the cloud (with $rk_{\mathbf{a}\rightarrow\mathbf{b}}$) and Bob (with $pk_{\mathbf{b}}$) collude together, they cannot compute the private key of Alice (i.e., $sk_{\mathbf{a}}$) due to the hardness of computing the DL problem in G_1.

Unfortunately, how to design such type of collusion-resistant proxy re-signature schemes while also sup-porting public auditing (i.e., blockless verifiability and non-malleability) remains to be seen. Essentially, since collusion-resistant proxy re-signature schemes generally have two levels of signatures (i.e., the first level is signed by a user and the second level is re-signed by the proxy), where the two levels of signatures are in different forms and need to be verified differently, achieving blockless verifiability on both of the two levels of signatures and verifying them together in a public auditing mechanism is challenging. We will leave this problem for our future work.

6. EXTENSION

In this section, we will utilize several different meth-ods to extend our mechanism in terms of detection probability, scalability and reliability.

6.1 Detection Probability

As presented in our mechanism, a verifier selects a number of random blocks instead of choosing all the blocks in shared data, which can improve the efficiency of auditing. Previous work [3] 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, referred to as sample strategies [3]. More specif-ically, 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. Further discussions and analyses about sample strategies can be found in [3].

To further reduce the number of the undetected pol-luted blocks in shared data and improve the detection probability, besides increasing the number of random selected blocks in one auditing task mentioned in the last paragraph, a verifier can also perform multiple auditing tasks on the same shared data. If the detection probability in a single auditing task is $P_{\mathbf{S}}$, then the total detection probability for a number of t multiple auditing
tasks is

$$P_{\mathbf{M}} = 1 - (1 - P_{\mathbf{S}})^{\mathbf{t}}.$$

For instance, if the detection probability in a single auditing task is $P_{\mathbf{S}} = 95\%$, then the total detection probability with two different auditing tasks on the same shared data is $P_{\mathbf{M}} =$ 99.75%. Note that to achieve a higher detection probability, both of the two methods require a verifier to spend more communication and computation cost during auditing.

the properties of bilinear maps. With batch auditing, a public verifier can perform multiple auditing tasks simultaneously. Compared to the batch auditing in [7], where the verification metadata (i.e, signatures) in each auditing task are generated by a single user, our batch auditing method needs to perform on multiple auditing tasks where the verification metadata in each auditing task are generated by a group of users. Clearly, designing batch auditing for our mechanism is more complicated and challenging than the one in [7].

More concretely, if the total number of auditing tasks received in a short time is t, then the size of the group for each task is $d_{\mathbf{j}}$, for $j \in [1, t]$, each auditing message is represented as $\{(l, \eta_{\mathbf{j|l}})\}_{\mathbf{l\in L}j}$, for $j \in [1, t]$, each au-diting proof is described as $\{\boldsymbol{\alpha}_{\mathbf{j}} , \boldsymbol{\beta}_{\mathbf{j}} , \{id_{\mathbf{j|l}}\}_{\mathbf{l\in L}j} \}$, where
$\boldsymbol{\alpha}_{\mathbf{j}} = (\alpha_{\mathbf{j}|1}, ..., \alpha_{\mathbf{j|d}j})$ and $\boldsymbol{\beta}_{\mathbf{j}} = (\beta_{\mathbf{j}|1}, ..., \beta_{\mathbf{j|d}j})$, for $j \in [1, t]$, and all the existing users' public keys for each group are denoted as $(pk_{\mathbf{j}|1}, ..., pk_{\mathbf{j|d}j})$, for $j \in [1, t]$. Based on the properties of bilinear maps, the public verifier can perform batch auditing as below.

6.2 Scalability

Now we discuss how to improve the scalability of our proposed mechanism by reducing the total number of re-signing keys in the cloud and enabling batch auditing for verifying multiple auditing tasks simultaneously.

Reduce the Number of Re-signing Keys. As de-scribed in, the cloud needs to establish and main-tain a re-signing key for each pair of two users in the group. Since the number of users in the group is denoted as d, the total number of re-signing keys for the group is d(d − 1)/2. Clearly, if the cloud data is shared by a very large number of users, e.g. d = 200, then the total number of re-signing keys that the cloud has to securely store and manage is 19, 900, which significantly increases the complexity of key management in cloud.

To reduce the total number of re-signing keys required in the cloud and improve the scalability of our mech-anism, the original user, who performs as the group manager, can keep a short priority list (PL) with only a small subset of users instead of the entire PL with all the users in the group. More specifically, if the total number of users in the group is still d = 200 and the size of a short PL is d' = 5, which means the cloud is able to convert signatures of a revoked user only into one of these five users shown in the short PL, then the total number of re-signing keys required with the short PL of 5 users is 990. It is only 5% of the number of re-signing keys with the entire PL of all the 200 users.

Batch Auditing for Multiple Auditing Tasks. In many cases, the public verifier may need to handle multiple auditing tasks in a very short time period. Clearly, asking the public verifier to perform these au-diting requests independently (one by one) may not be efficient. Therefore, to improve the scalability of our public auditing mechanism in such cases, we can further extend to support batch auditing [7] by utilizing (3) where $\theta_j \in Z^*_p$, for $j \in [1, t]$, is a random chosen by the public verifier. The correctness of the above equation is based on all the t auditing proofs are correct. The left hand side (LHS) of this equation can be expended as according to the security analysis of batch verification in [25], the probability that the public verifier accepts an invalid auditing proof with batch auditing is 1/p (since randoms (θ_1, ..., θ_t) are elements of Z_p), which is negligible. Therefore, if the above equation holds, then the public verifier believes all the t auditing proofs are correct.

One of the most important advantages of batch au-diting is that it is able to reduce the total number of pairing operations, which are the most time consuming operations during verification. According to Equation (3), batch auditing can reduce the total number of pairing operations for t auditing tasks to td + 1, while verifying these t auditing tasks independently requires td + t pairing operations. Moreover, if all the t auditing tasks are all from the same group, where the size of the group is d and all the existing users public keys for the group are (pk_1, ..., pk_d), then batch auditing on t auditing tasks can be further optimized as follows In this case, the total number of pairing operations during batch auditing can be significantly reduced to

6.3 Reliability

In our mechanism, it is very important for the cloud to securely store and manage the re-signing keys of the group, so that the cloud can correctly and successfully convert signatures from a revoked user to an existing user when it is necessary. However, due to the existence of internal attacks, simply storing these re-signing keys in the cloud with a single re-signing proxy may some-times allow inside attackers to disclose these re-signing keys and arbitrarily convert signatures on shared data, even no user is revoking from the group. Obviously, the arbitrary misuse of re-signing keys will change the own-ership of corresponding blocks in shared data without users' permission, and affect the integrity of shared data in the cloud. To prevent the arbitrary use of re-signing keys and enhance the reliability of our mechanism, we propose an extended version of our mechanism, denoted as, in the multi-proxy model.

By leveraging an (s, t)-Shamir Secret Sharing ($s \geq 2t - 1$) [18] and s multiple proxies, each re-signing key is divided into s pieces and each piece is distributed to one proxy. These multiple proxies belong to the same cloud, but store and manage each piece of a re-signing key independently (as described in Fig. 8). Since the cloud needs to store keys and data separately [23], the cloud also has another server to store shared data and corresponding signatures. In each proxy is able to convert signatures with its own piece, and as long as t or more proxies (the majority) are able to correctly convert signatures when user revocation happens, the cloud can successfully convert signatures from a revoked user to an existing user. Similar multi-proxy model was also recently used in the cloud to secure the privacy of data with re-encryption techniques [26].

According to the security properties of an (s, t)-Shamir Secret Sharing, even up to $t-1$ proxies are compromised by an inside attacker, it is still not able to reveal a re-signing key or arbitrarily transform signatures on shared data. For most of algorithms are as the same as in, except the two algorithms for generating re-signing keys and re-signing signatures, denoted as **ReKey** and **ReSign** respectively. We use to distin-guish them from the corresponding algorithms in the single proxy model.

7. OVERHEAD ANALYSIS

In this section, we evaluate the performance of our mechanism in experiments. We utilize Pairing Based Cryptography (PBC) Library [27] to implement crypto-graphic operations in our mechanism. All the experi-ments 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 [4], [14], the size of each block can be set as 2 KB, then the total size of shared data is 2 GB.

As introduced in Section 1, the main purpose of is to improve the efficiency of user revocation. With our mechanism, the cloud is able to re-sign blocks for existing users during user revocation, so that an existing user does not need to download blocks and re-compute signatures by himself/herself. In contrast, to revoke a user in the group with the straightforward method extended from previous solutions, 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.

8.RELATED WORK

Provable Data Possession (PDP), first proposed by Ateniese *et al.* [3], allows a public verifier to check the correctness of a client's data stored at an untrusted server. By utilizing RSA-based homomorphic authenti-cators and sampling strategies, the verifier is able to publicly audit the integrity of data without retrieving the entire data, which is referred to as public verifiability or public auditing. Shacham and Waters [4] designed

nication overhead in the phase of data repair, Chen *et al.* [33] 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.* [11] constructed an LT code-based secure cloud storage mechanism. Compared to previous mechanisms [5], [33], this mechanism can avoid high decoding computation costs for data users and save com-putation resources for online data owners during data repair. Recently, Wang *et al.* [34] proposed a certificateless public auditing mechanism to reduce security risks in certificate management compared to previous certificate-based solutions.

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.* [7] 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 [14] first proposed a mecha-nism 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 [16] is designed to preserve identity privacy for a large number of users. However, it fails to support public auditing.

Proofs of Retrievability (POR) [35] is another direction to check the correctness of data stored in a semi-trusted server. Unfortunately, POR and its subsequent work [36] do not support public verification, which fails to satisfy the design objectives in our paper.

9.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 semi-trusted 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.

REFERENCES

[1] B. Wang, B. Li, and H. Li, "Public Auditing for Shared Data with Efficient User Revoation in the Cloud," in *the Proceedings of IEEE INFOCOM 2013*, 2013, pp. 2904–2912.

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

[3] 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.

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

[5] 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.

[6] 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.

[7] 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.

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

[9] C. Wang, Q. Wang, K. Ren, and W. Lou, "Towards Secure and Dependable Storage Services in Cloud Computing," *IEEE Transactions on Services Computing*, vol. 5, no. 2, pp. 220–232, 2011.

[10] Y. Zhu, G.-J. Ahn, H. Hu, S. S. Yau, H. G. An, and S. Chen, "Dynamic Audit Services for Outsourced Storage in Clouds," *IEEE Transactions on Services Computing*, accepted.

[11] 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.

[12] J. Yuan and S. Yu, "Proofs of Retrievability with Public Verifiabil-ity and Constant Communication Cost in Cloud," in *Proceedings of ACM ASIACCS-SCC'13*, 2013.

[13] H. Wang, "Proxy Provable Data Possession in Public Clouds," *IEEE Transactions on Services Computing*, accepted.

[14] 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.

[15] S. R. Tate, R. Vishwanathan, and L. Everhart, "Multi-user Dy-namic Proofs of Data Possession Using Trusted Hardware," in *Proceedings of ACM CODASPY'13*, 2013, pp. 353–364.

[16] 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.

[17] 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.

[18] A. Shamir, "How to share a secret," in *Communication of ACM*, vol. 22, no. 11, 1979, pp. 612–613.

[19] B. Wang, H. Li, and M. Li, "Privacy-Preserving Public Auditing for Shared Cloud Data Supporting Group Dynamics," in *the Proceedings of IEEE ICC 2013*, 2013.

[20] B. Wang, S. S. Chow, M. Li, and H. Li, "Storing Shared Data on the Cloud via Security-Mediator," in *Proceedings of IEEE ICDCS 2013*, 2013.

[21] M. Li, N. Cao, S. Yu, and W. Lou, "FindU: Private-Preserving Personal Profile Matching in Mobile Social Networks," in *Proceedings of IEEE INFOCOM*, 2011, pp. 2435 – 2443.

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

[23] M. van Dijk, A. Juels, A. Oprea, R. L. Rivest, E. Stefanov, and N. Triandopoulos, "Hourglass schemes: how to prove that cloud files are encrypted," in *the Proceedings of ACM CCS 2012*, 2012, pp. 265–280.

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

[25] A. L. Ferrara, M. Green, S. Hohenberger, and M. Ø. Pedersen, "Practical Short Signature Batch Verification," in *Proc. CT-RSA*. Springer-Verlag, 2009, pp. 309–324.

[26] L. Xu, X. Wu, and X. Zhang, "CL-PRE: a Certificateless Proxy Re-Encryption Scheme for Secure Data Sharing with Public Cloud," in *the Proceedings of ACM ASIACCS 2012*, 2012.

[27] Pairing Based Cryptography (PBC) Library. [Online]. Available: http://crypto.stanford.edu/pbc/

[28] The Java Pairing Based Cryptography (jPBC) Library Benchmark. [Online]. Available: http://gas.dia.unisa.it/projects/jpbc/benchmark.html

[29] J. Yuan and S. Yu. Efficient Public Integrity Checking for Cloud Data Sharing with Multi-User Modification. [Online]. Available: http://eprint.iacr.org/2013/484

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

[31] 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.

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

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

[34] B. Wang, B. Li, and H. Li, "Certificateless Public Auditing for Data Integrity in the Cloud," in *Proceedings of IEEE CNS 2013*, 2013, pp. 276–284.

[35] A. Juels and B. S. Kaliski, "PORs: Proofs pf Retrievability for Large Files," in *Proceedings of ACM CCS'07*, 2007, pp. 584–597.

[36] D. Cash, A. Kupcu, and D. Wichs, "Dynamic Proofs of Retrievability via Oblivious RAM," in *Proceedings of EUROCRYPT 2013*, 2013, pp. 279–295.