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

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

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. In this paper, we propose 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, 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. Moreover, our mechanism is able to support batch auditing by verifying multiple auditing tasks simultaneously. Experimental results show that our mechanism can significantly improve the efficiency of user revocation.

Index Terms—Public auditing, shared data, user revocation, cloud computing.

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 Possession [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, 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 [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 revocation (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 resources by downloading and verifying blocks, and by

- *Boyang Wang and Hui Li are with the State Key Laboratory of Integrated Service Networks, Xidian University, Xi'an, Shaanxi, 710071, China. E-mail: bywang@mail.xidian.edu.cn; lihui@mail.xidian.edu.cn.*
- *Baochun Li is with the Department of Electrical and Computer Engineering, University of Toronto, Toronto, ON, M5S 3G4, Canada. E-mail: bli@eecg.toronto.edu.*
- *This work is supported by the NSF of China (No. 61272457 and 61170251), Fundamental Research Funds for the Central Universities (No. K50511010001), National 111 Program (No. B08038), Doctoral Foundation of Ministry of Education of China (No. 20100203110002), Program for Changjiang Scholars and Innovative Research Team in University (PCSIRT 1078).*
- *Most part of this work was done at University of Toronto. A preliminary version [1] of this paper is in Proceedings of the 32nd IEEE International Conference on Computer Communications (IEEE INFOCOM 2013).*

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 this matter even worse, existing users may access their data sharing services provided by the cloud with resource-limited devices, such as mobile phones, which further prevents existing users from maintaining the correctness of shared data efficiently during user revocation.

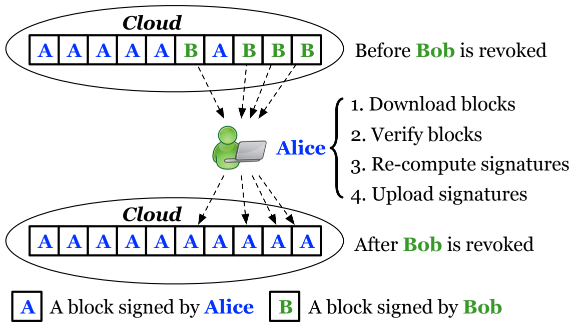


Fig. 1. Alice and Bob share data in the cloud. When Bob is revoked, Alice re-signs the blocks that were previously signed by Bob with her private key.

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 Panda, a novel public auditing mechanism for the integrity of shared data with efficient user revocation in the cloud. In our mechanism, by utilizing the idea of proxy re-signatures [17], 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 (described in Section 4), which traditional proxy re-signatures do no have, our mechanism is always able to check the integrity of shared data without retrieving the entire data from the cloud.

Moreover, our proposed mechanism is scalable, which

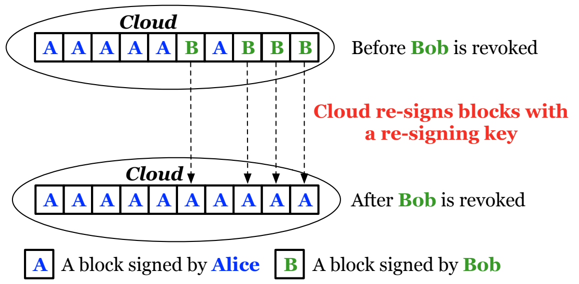


Fig. 2. When Bob is revoked, the cloud re-signs the blocks that were previously signed by Bob with a re-signing key.

indicates it is not only able to efficiently support a large number of users to share data and but also able to handle 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 preliminaries 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 proposed 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

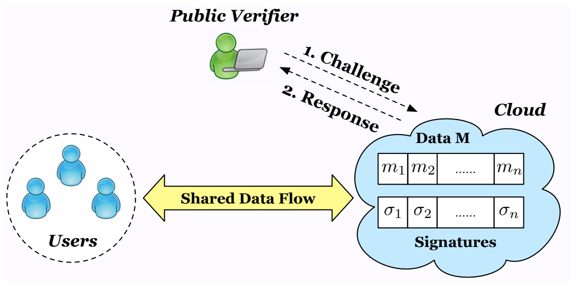


Fig. 3. The system model includes the cloud, the public verifier, and users.

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. Considering these factors, 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. Specifically, 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 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 following 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 preliminaries, including bilinear maps, security assumptions, homomorphic authenticators and proxy re-signatures.

3.1 Bilinear Maps

Let $\mathbb{G}_1$ and $\mathbb{G}_2$ be two multiplicative cyclic groups of prime order p, g be a generator of $\mathbb{G}_1$. Bilinear map e is a map e: $\mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_2$ with the following properties: 1) **Computability**: there exists an efficient algorithm for computing map e. 2) **Bilinearity**: for all $u, v \in \mathbb{G}_1$ and $a, b \in \mathbb{Z}_p$, $e(u^a, v^b) = e(u, v)^{ab}$. 3) **Non-degeneracy**: $e(g, g) \neq 1$.

3.2 Security Assumptions

The security of our mechanism is based on the following security assumptions.

Computational Diffie-Hellman (CDH) Problem. Let $a, b \in \mathbb{Z}_p^*$, given $g, g^a, g^b \in \mathbb{G}_1$ as input, output $g^{ab} \in \mathbb{G}_1$.

Definition 1: **Computational Diffie-Hellman (CDH) Assumption.** For any *probabilistic polynomial time* adversary $\mathcal{A}_{\mathcal{CDH}}$, the advantage of adversary $\mathcal{A}_{\mathcal{CDH}}$ on solving the CDH problem in $\mathbb{G}_1$ is negligible, which is defined as

$$Pr[\mathcal{A}_{\mathcal{CDH}}(g, g^a, g^b) = (g^{ab}) : a, b \stackrel{R}{\leftarrow} \mathbb{Z}_p^*] \leq \epsilon.$$

For the ease of understanding, we can also say computing the CDH problem in $\mathbb{G}_1$ is computationally infeasible or hard under the CDH assumption.

Discrete Logarithm (DL) Problem. Let $a \in \mathbb{Z}_p^*$, given $g, g^a \in \mathbb{G}_1$ as input, output a.

Definition 2: **Discrete Logarithm (DL) Assumption.** For any *probabilistic polynomial time* adversary $\mathcal{A}_{\mathcal{DL}}$, the advantage of adversary $\mathcal{A}_{\mathcal{DL}}$ on solving the DL problem in $\mathbb{G}_1$ is negligible, which is defined as

$$Pr[\mathcal{A}_{\mathcal{DL}}(g, g^a) = (a) : a \stackrel{R}{\leftarrow} \mathbb{Z}_p^*] \leq \epsilon.$$

Similarly, we can also say computing the DL problem in $\mathbb{G}_1$ is computationally infeasible or hard under the DL assumption.

3.3 Homomorphic Authenticators

Homomorphic authenticators [3], 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 as building blocks in the previous public auditing mechanisms [3]–[15], [19], [20]. 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 $(\mathtt{pk}, \mathtt{sk})$ denote the signer's public/private key pair, σ_1 denote the signature on block $m_1 \in \mathbb{Z}_p$, and σ_2 denote the signature on block $m_2 \in \mathbb{Z}_p$.

- ***Blockless verifiability:*** Given σ_1 and σ_2, two random values α_1, α_2 in $\mathbb{Z}_p$ and a block $m' = \alpha_1 m_1 + \alpha_2 m_2 \in \mathbb{Z}_p$, 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 $\mathbb{Z}_p$ and a block $m' = \alpha_1 m_1 + \alpha_2 m_2 \in \mathbb{Z}_p$, a user, who does not have private key $\mathtt{sk}$, 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 via a challenge-and-response protocol, while the entire data does not need to be downloaded 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 Proxy Re-signatures

Proxy re-signatures, first proposed by Blaze *et al.* [17], 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 during user revocation.

3.5 Shamir Secret Sharing

An (s,t)-Shamir Secret Sharing scheme [18] ($s \geq 2t-1$), 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

$$f(x) = a_{t-1}x^{t-1} + \cdots + a_1 x + a_0,$$

where $a_{t-1}, ..., a_1 \overset{R}{\leftarrow} \in \mathbb{Z}_p^*$. Then, the secret is $\pi = a_0$, and each piece of this secret is actually a point of polynomial $f(x)$, i.e. $(x_i, f(x_i))$, for $1 \leq i \leq s$. The secret π can be recovered by any t points of this $t-1$ degree polynomial $f(x)$ with Lagrange polynomial interpolation. Shamir Secret Sharing is widely used in key management schemes [18] and secure multi-party computation [21].

4 A New Proxy Re-signature Scheme

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.

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 mechanism, 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 algorithms: **KeyGen**, **ReKey**, **Sign**, **ReSign** and **Verify**. Details 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

$$e(\sigma, g) = e((H(id)w^m)^a, g) = e(H(id)w^m, \mathtt{pk}_A).$$

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 defined in the previous proxy re-signature scheme [22], 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.

External Security: We show that if a (t', ϵ')-algorithm $\mathcal{A}$, operated by an external adversary, can generate a forgery of a signature under HAPS with the time of t' and advantage of ϵ' after making at most q_H hash queries, q_S signing queries, q_R re-signing queries, and requesting at most q_K public keys, then there exists a (t, ϵ)-algorithm $\mathcal{B}$ that can solve the CDH problem in $\mathbb{G}_1$ with

$$t \leq t' + q_H c_{\mathbb{G}_1} + q_S c_{\mathbb{G}_1} + 2q_R c_P,$$
$$\epsilon \geq \epsilon' / q_H q_K,$$

where one exponentiation on $\mathbb{G}_1$ takes time $c_{\mathbb{G}_1}$ and one pairing operation takes time c_P. Specifically, on input (g, g^a, g^b), the CDH algorithm $\mathcal{B}$ simulates a proxy re-signature security game for algorithm $\mathcal{A}$ as follows:

Public Keys: As $\mathcal{A}$ requests the creation of system users, $\mathcal{B}$ guesses which one $\mathcal{A}$ will attempt a forgery against. Without loss of generality, we assume the target public key as $\mathtt{pk}_v$ and set it as $\mathtt{pk}_v = g^a$. For all other

Scheme Details: Let $\mathbb{G}_1$ and $\mathbb{G}_2$ be two groups of order p, g be a generator of $\mathbb{G}_1$, $e : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_2$ be a bilinear map, w be another generator of $\mathbb{G}_1$. The global parameters are $(e, p, \mathbb{G}_1, \mathbb{G}_2, g, w, H)$, where H is a hash function with $H : \{0,1\}^* \rightarrow \mathbb{G}_1$.

KeyGen. Given global parameters $(e, p, \mathbb{G}_1, \mathbb{G}_2, g, w, H)$, a user u_A selects a random $a \in \mathbb{Z}_p^*$, and outputs public key $\mathtt{pk}_A = g^a$ and private key $\mathtt{sk}_A = a$.

ReKey. The proxy generates a re-signing key $\mathtt{rk}_{A\rightarrow B}$ as follows: (1) the proxy generates a random $r \in \mathbb{Z}_p^*$ and sends it to user u_A; (2) user u_A computes and sends r/a to user u_B, where $\mathtt{sk}_A = a$; (3) user u_B calculates and sends rb/a to the proxy, where $\mathtt{sk}_B = b$; (4) the proxy recovers $\mathtt{rk}_{A\rightarrow B} = b/a \in \mathbb{Z}_p^*$.

Sign. Given private key $\mathtt{sk}_A = a$, block $m \in \mathbb{Z}_p$ and block identifier id, user u_A outputs the signature on block m as:

$$\sigma = (H(id)w^m)^a \in \mathbb{G}_1.$$

ReSign. Given re-signing key $\mathtt{rk}_{A\rightarrow B}$, public key $\mathtt{pk}_A$, signature σ, block $m \in \mathbb{Z}_p$ and block identifier id, the proxy checks that **Verify**$(\mathtt{pk}_A, m, id, \sigma) \overset{?}{=} 1$. If the verification result is 0, the proxy outputs $\perp$; otherwise, it outputs

$$\sigma' = \sigma^{\mathtt{rk}_{A\rightarrow B}} = (H(id)w^m)^{a\cdot b/a} = (H(id)w^m)^b.$$

Verify. Given public key $\mathtt{pk}_A$, block m, block identifier id, and signature σ, a verifier outputs 1 if

$$e(\sigma, g) = e(H(id)w^m, \mathtt{pk}_A),$$

and 0 otherwise.

Fig. 4. Details of HAPS.

public keys, we set $\mathtt{pk}_i = g^{x_i}$ for a random $x_i \in \mathbb{Z}_p^*$. The total number of public keys requested is q_K.

Oracle Queries: There are three types of oracle queries that $\mathcal{B}$ must answer: hash query $\mathcal{Q}_{hash}$, signing query $\mathcal{Q}_{sign}$ and re-signing query $\mathcal{Q}_{resign}$.

For each hash query $\mathcal{Q}_{hash}$ on input (id_i, m_i), check if there is an entry in table T_H. If so, output the corresponding value; otherwise guess if (id_i, m_i) is the identifier id^* and block m^* that $\mathcal{A}$ will attempt to use in a forgery. If $id_i = id^*$ and $m_i = m^*$, output $H(id_i)w^{m_i} = g^b$; otherwise, select a random $y_i \in \mathbb{Z}_p^*$ and output $H(id_i)w^{m_i} = g^{y_i}$. Record (id_i, m_i, y_i) in the table T_H for each $id_i \neq id^*$ or $m_i \neq m^*$.

For each signing query $\mathcal{Q}_{sign}$ on input $(\mathtt{pk}_j, id_i, m_i)$, if $j \neq v$, output the signature as $\sigma = (H(id_i)w^{m_i})^{x_j}$ (via calling hash query). If $j = v$ and $id_i \neq id^*$, or $j = v$ and $m_i \neq m^*$, return the signature as $\sigma = (g^{y_i})^a = (g^a)^{y_i}$; otherwise, abort.

For each re-signing query $\mathcal{Q}_{resign}$ on input $(\mathtt{pk}_i, \mathtt{pk}_j, id_k, m_k, \sigma)$, if **Verify**$(\mathtt{pk}_i, m_k, id_k, \sigma)\neq 1$, output $\perp$. Otherwise, output $\mathcal{Q}_{sign}(\mathtt{pk}_j, id_k, m_k)$ (via calling signing query).

Forgery: Eventually $\mathcal{A}$ outputs a forgery $(\mathtt{pk}_j, id, m, \sigma)$. If $v \neq j$, then $\mathcal{B}$ guessed the wrong target user and must abort. If **Verify**$(\mathtt{pk}_j, m, id, \sigma)\neq 1$ or (id, m, σ) is the result of any $\mathcal{Q}_{sign}$ or $\mathcal{Q}_{resign}$, $\mathcal{B}$ also aborts. Otherwise, $\mathcal{B}$ outputs $\sigma = (H(id)w^m)^a = (g^b)^a = g^{ab}$ as the proposed CDH problem.

The probability that $\mathcal{B}$ will guess the target user correctly is $1/q_K$, and the probability that $\mathcal{B}$ will guess the forged block identifier id^* and forged block m^* is $1/q_H$. Therefore, if $\mathcal{A}$ generates a forgery of a signature with probability ϵ', then $\mathcal{B}$ solves the CDH problem with probability $\epsilon'/q_H q_K$. Algorithm $\mathcal{B}$ requires one exponentiation on $\mathbb{G}_1$ for each hash query, one extra exponentiation on $\mathbb{G}_1$ for each signing query and two extra pairing operations for each re-signing query, so its running time is $\mathcal{A}$'s running time plus $q_H c_{\mathbb{G}_1} + q_S c_{\mathbb{G}_1} + 2q_R c_P$.

Internal Security: We now prove that, if a (t', ϵ')-algorithm $\mathcal{A}$, operated by the proxy, can generate a forgery of a signature with the time of t' and advantage of ϵ' 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 $\mathbb{G}_1$ with

$$t \leq t' + q_H c_{\mathbb{G}_1} + q_S c_{\mathbb{G}_1}, \quad \epsilon \geq \epsilon'/q_H q_K.$$

On input (g, g^a, g^b), the CDH algorithm $\mathcal{B}$ simulates a proxy re-signature security game for algorithm $\mathcal{A}$ as follows:

Public Keys: Without loss of generality, we set the target key as $\mathtt{pk}_v = g^a$. For each key $i \neq v$, choose a random $x_i \in \mathbb{Z}_p^*$, and set $\mathtt{pk}_i = (g^a)^{x_i}$. The total number of public keys requested is q_K.

Oracle Queries: There are three types of queries that $\mathcal{B}$ must answer: hash query $\mathcal{Q}_{hash}$, signing query $\mathcal{Q}_{sign}$ and rekey query $\mathcal{Q}_{rekey}$.

For each hash query $\mathcal{Q}_{hash}$ on input (id_i, m_i), check if there is an entry in table T_H. If so, output the corresponding value; otherwise guess if (id_i, m_i) is the identifier id^* and block m^* that $\mathcal{A}$ will attempt to use in a forgery. If $id_i = id^*$ and $m_i = m^*$, output $H(id_i)w^{m_i} = g^b$; otherwise, select a random $y_i \in \mathbb{Z}_p^*$ and output $H(id_i)w^{m_i} = g^{y_i}$. Record (id_i, m_i, y_i) in the table T_H for each $id_i \neq id^*$ or $m_i \neq m^*$.

For each signing query $\mathcal{Q}_{sign}$ on input $(\mathtt{pk}_j, id_i, m_i)$, if $id_i \neq id^*$ or $m_i \neq m^*$, return the signature as $\sigma = (\mathtt{pk}_j)^{y_i}$ (via calling hash query); else abort.

For each rekey query $\mathcal{Q}_{rekey}$ on input $(\mathtt{pk}_i, \mathtt{pk}_j)$, if $i = v$ or $j = v$, abort; else return $\mathtt{rk}_{i\rightarrow j} = (x_j/x_i)$.

Forgery: Eventually $\mathcal{A}$ outputs a forgery $(\mathtt{pk}_j, id, m, \sigma)$. If $v \neq j$, then $\mathcal{B}$ guessed the wrong target user and must abort. If **Verify**$(\mathtt{pk}_j, m, id, \sigma)\neq 1$ or (id, m, σ) is the result of any $\mathcal{Q}_{sign}$, $\mathcal{B}$ also aborts. Otherwise, $\mathcal{B}$ outputs $\sigma = (H(id)w^m)^a = (g^b)^a = g^{ab}$ as the proposed CDH problem.

Similarly as external security, the probability that $\mathcal{B}$ will guess the target user correctly is $1/q_K$, and the probability that $\mathcal{B}$ will guess the forged block identifier id^* and forged block m^* is $1/q_H$. Therefore, if $\mathcal{A}$ generates a forgery of a signature with probability ϵ', then $\mathcal{B}$ solves the CDH problem with probability $\epsilon'/q_H q_K$. Algorithm $\mathcal{B}$ requires one exponentiation on $\mathbb{G}_1$ for each hash query, one extra exponentiation on $\mathbb{G}_1$ for each signing query, so its running time is $\mathcal{A}$'s running time plus $q_H c_{\mathbb{G}_1} + q_S c_{\mathbb{G}_1}$.

According to what we analyzed above, if the advantage of an adversary for generating a forgery of a signature under the external or internal security game is non-negligible, then we can find an algorithm to solve the CDH problem in $\mathbb{G}_1$ with a non-negligible probability, which contradicts to the assumption that the CDH problem is computationally infeasible in $\mathbb{G}_1$. Therefore, it is computationally infeasible to generate a forgery of a signature in HAPS under the CDH assumption. □

***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 $\mathtt{pk}_A$, two random numbers y_1, $y_2 \in \mathbb{Z}_p^*$, 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

$$e(\sigma_1^{y_1} \cdot \sigma_2^{y_2}, g) \overset{?}{=} e(H(id_1)^{y_1} H(id_2)^{y_2} w^{m'}, \mathtt{pk}_A), \quad (1)$$

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:

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

It is clear that HAPS can support blockless verifiability.

Non-malleability. Meanwhile, an adversary, who does not have private key $\mathtt{sk}_A = a$, 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, 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 PANDA

5.1 Overview

Based on the new proxy re-signature scheme and its properties in the previous section, we now present Panda — a public auditing mechanism for shared data with efficient user revocation. In our mechanism, the original user acts as the group manager, who is able to revoke users from the group when it is necessary. Meanwhile, we allow the cloud to perform as the semi-trusted proxy and translate signatures for users in the group with re-signing keys. As emphasized in recent work [23], for security reasons, it is necessary for the cloud service providers to storage data and keys separately on different servers inside the cloud in practice. Therefore, in our mechanism, we assume the cloud has a server to store shared data, and has another server to manage re-signing keys. To ensure the privacy of cloud shared data at the same time, additional mechanisms, such as [24], can be utilized. The details of preserving data privacy are out of scope of this paper. The main focus of this paper is to audit the integrity of cloud shared data.

5.2 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 (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.

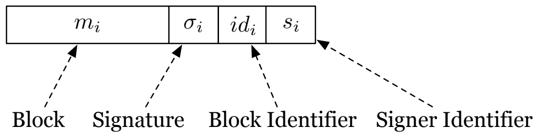


Fig. 6. Each block is attached with a signature, a block identifier and a signer identifier.

By leveraging index hash tables [8], [14], we allow a user to modify a single block efficiently without changing 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.

5.3 Construction of Panda

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

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. As argued in previous section, we still assume that private channels exist between each pair of entities

Scheme Details: Let $\mathbb{G}_1$ and $\mathbb{G}_2$ be two groups of order p, g be a generator of $\mathbb{G}_1$, $e : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_2$ be a bilinear map, w be another generator of $\mathbb{G}_1$. The global parameters are $(e, p, \mathbb{G}_1, \mathbb{G}_2, g, w, H)$, where H is a hash function with $H : \{0,1\}^* \rightarrow \mathbb{G}_1$. 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. User u_i generates a random $\pi_i \in \mathbb{Z}_p^*$, and outputs public key $\mathtt{pk}_i = g^{\pi_i}$ and private key $\mathtt{sk}_i = \pi_i$. 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 $\mathtt{rk}_{i\rightarrow j}$ as follows: (1) the cloud generates a random $r \in \mathbb{Z}_p$ and sends it to user u_i; (2) user u_i sends r/π_i to user u_j, where $\mathtt{sk}_i = \pi_i$; (3) user u_j sends $r\pi_j/\pi_i$ to the cloud, where $\mathtt{sk}_j = \pi_j$; (4) the cloud recovers $\mathtt{rk}_{i\rightarrow j} = \pi_j/\pi_i \in \mathbb{Z}_p^*$.

Sign. Given private key $\mathtt{sk}_i = \pi_i$, block $m_k \in \mathbb{Z}_p$ and its block identifier id_k, where $k \in [1, n]$, user u_i outputs the signature on block m_k as:

$$\sigma_k = (H(id_k)w^{m_k})^{\pi_i} \in \mathbb{G}_1.$$

ReSign. Given re-signing key $\mathtt{rk}_{i\rightarrow j}$, public key $\mathtt{pk}_i$, signature σ_k, block m_k and block identifier id_k, the cloud first checks that $e(\sigma_k, g) \overset{?}{=} e(H(id_k)w^{m_k}, \mathtt{pk}_i)$. If the verification result is 0, the cloud outputs $\perp$; otherwise, it outputs

$$\sigma'_k = \sigma_k^{\mathtt{rk}_{i\rightarrow j}} = (H(id_k)w^{m_k})^{\pi_i \cdot \pi_j/\pi_i} = (H(id_k)w^{m_k})^{\pi_j}.$$

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

ProofGen. A public verifier 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 $\eta_l \in \mathbb{Z}_q^*$, for $l \in L$ and q is a much smaller prime than p.
3) Outputs an auditing message $\{(l, \eta_l)\}_{l\in L}$, 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) According to the signer identifier of each selected block, 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. 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 subset L_i, the cloud computes $\alpha_i = \sum_{l\in L_i} \eta_l m_l \in \mathbb{Z}_p$ and $\beta_i = \prod_{l\in L_i} \sigma_l^{\eta_l} \in \mathbb{G}_1$.
3) The cloud outputs an auditing proof $\{\boldsymbol{\alpha}, \boldsymbol{\beta}, \{id_l, s_l\}_{l\in L}\}$, and sends it to the verifier, where $\boldsymbol{\alpha} = (\alpha_1, ..., \alpha_d)$ and $\boldsymbol{\beta} = (\beta_1, ..., \beta_d)$.

ProofVerify. Given an auditing message $\{(l, \eta_l)\}_{l\in L}$, an auditing proof $\{\boldsymbol{\alpha}, \boldsymbol{\beta}, \{id_l, s_l\}_{l\in L}\}$, and all the existing users' public keys $(\mathtt{pk}_1, ..., \mathtt{pk}_d)$, the public verifier checks the correctness of this auditing proof as

$$e(\prod_{i=1}^{d} \beta_i, g) \overset{?}{=} \prod_{i=1}^{d} e(\prod_{l\in L_i} H(id_l)^{\eta_l} \cdot w^{\alpha_i}, \mathtt{pk}_i). \qquad (2)$$

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

Fig. 5. Details of Panda.

during the generation of re-signing keys, and there is no collusion. 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 verification on data integrity is performed via a challenge-and-response protocol between the cloud and a public verifier. More specifically, the cloud is able to generate a proof of possession of shared data in **ProofGen** under the challenge of a public verifier. In **ProofVerify**, a public verifier is able to check the correctness of a proof responded by the cloud.

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 ask the original user to create a priority list (PL). Every existing user's id is in the PL and listed in the order of re-signing 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 private key of the original user (i.e., the group manager).

Based on the properties of bilinear maps, the correctness of our mechanism in **ProofVerify** can be explained as follows.

$$\begin{aligned}
e(\prod_{i=1}^{d} \beta_i, g) &= \prod_{i=1}^{d} e(\prod_{l\in L_i} \sigma_l^{\eta_l}, g) \\
&= \prod_{i=1}^{d} e(\prod_{l\in L_i} (H(id_l)w^{m_l})^{\pi_i \eta_l}, g) \\
&= \prod_{i=1}^{d} e(\prod_{l\in L_i} H(id_l)^{\eta_l} \cdot \prod_{l\in L_i} w^{m_l \eta_l}, g^{\pi_i}) \\
&= \prod_{i=1}^{d} e(\prod_{l\in L_i} H(id_l)^{\eta_l} \cdot w^{\alpha_i}, \mathtt{pk}_i).
\end{aligned}$$

5.4 Security Analysis of Panda

***Theorem** 3: For the cloud, it is computationally infeasible to generate a forgery of an auditing proof in Panda as long as the DL assumption holds.*

Proof: Following the security game defined in [4], [14], 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 DL problem in $\mathbb{G}_1$ with a probability of $1 - 1/p$, which contradicts to the DL assumption (the advantage of solving DL problem $\mathbb{G}_1$ should be negligible). Specifically, we define Game 1 as follows:

Game 1: The public verifier sends an auditing message $\{(l, \eta_l)\}_{l \in L}$ 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 (2). However, the cloud generates a proof on incorrect shared data M' as $\{\boldsymbol{\alpha}', \boldsymbol{\beta}, \{id_l\}_{l \in L}\}$, where $\boldsymbol{\alpha}' = (\alpha_1, ..., \alpha_d)$, $\alpha'_i = \sum_{l \in L_i} \eta_l m'_l$, 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 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

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

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

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

Based on the properties of bilinear maps, we learn that

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

Because $\mathbb{G}_1$ is a cyclic group, then for two elements $u, v \in \mathbb{G}_1$, there exists $x \in \mathbb{Z}_p$ that $v = u^x$. Without loss of generality, given u, v, each w^{π_i} can generated as $w^{\pi_i} = u^{\xi_i} v^{\gamma_i} \in \mathbb{G}_1$, where ξ_i and γ_i are random values of $\mathbb{Z}_p$. 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 DL problem. Given $u, v = u^x \in \mathbb{G}_1$, 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 $\mathbb{Z}_p$, 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 $\mathbb{G}_1$. $\square$

5.5 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.

5.6 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 $\mathtt{sk}_b$) is able to collude with the cloud, who possesses a re-signing key (e.g., $\mathtt{rk}_{a \to b} = \mathtt{sk}_a / \mathtt{sk}_b$, then the cloud and Bob together are able to easily reveal the private key of an existing user (e.g., Alice's private key $\mathtt{sk}_a$).

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 $\mathtt{rk}_{a \to b} = \mathtt{pk}_b^{\mathtt{sk}_a}$ by Alice, then even the cloud (with $\mathtt{rk}_{a \to b}$) and Bob (with $\mathtt{pk}_b$) collude together, they cannot compute the private key of Alice (i.e., $\mathtt{sk}_a$) due to the hardness of computing the DL problem in $\mathbb{G}_1$.

Unfortunately, how to design such type of collusion-resistant proxy re-signature schemes while also supporting 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 OF PANDA

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

6.1 Detection Probability of Panda

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 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. Further discussions and analyses about sample strategies can be found in [3].

To further reduce the number of the undetected polluted 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_S, then the total detection probability for a number of t multiple auditing tasks is

$$P_M = 1 - (1 - P_S)^t.$$

For instance, if the detection probability in a single auditing task is $P_S = 95\%$, then the total detection probability with two different auditing tasks on the same shared data is $P_M = 99.75\%$. 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.

6.2 Scalability of Panda

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 described in Panda, the cloud needs to establish and maintain 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 mechanism, 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 auditing 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 Panda to support batch auditing [7] by utilizing 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_j, for $j \in [1,t]$, each auditing message is represented as $\{(l, \eta_{j|l})\}_{l \in L_j}$, for $j \in [1,t]$, each auditing proof is described as $\{\boldsymbol{\alpha}_j, \boldsymbol{\beta}_j, \{id_{j|l}\}_{l \in L_j}\}$, where $\boldsymbol{\alpha}_j = (\alpha_{j|1}, ..., \alpha_{j|d_j})$ and $\boldsymbol{\beta}_j = (\beta_{j|1}, ..., \beta_{j|d_j})$, for $j \in [1,t]$, and all the existing users' public keys for each group are denoted as $(\mathtt{pk}_{j|1}, ..., \mathtt{pk}_{j|d_j})$, for $j \in [1,t]$. Based on the properties of bilinear maps, the public verifier can perform batch auditing as below

$$e(\prod_{j=1}^{t}\prod_{i=1}^{d_j}\beta_{j|i}^{\theta_j}, g) \stackrel{?}{=} \prod_{j=1}^{t}\prod_{i=1}^{d_j} e(\prod_{l \in L_{j|i}} H(id_{j|l})^{\eta_{j|l}} \cdot w^{\alpha_{j|i}}, \mathtt{pk}_{j|i})^{\theta_j}, \tag{3}$$

where $\theta_j \in \mathbb{Z}_p^*$, 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

$$\begin{aligned} \mathtt{LHS} &= \prod_{j=1}^{t} e(\prod_{i=1}^{d_j}\beta_{j|i}, g)^{\theta_j} \\ &= \prod_{j=1}^{t}\prod_{i=1}^{d_j} e(\prod_{l \in L_{j|i}} H(id_{j|l})^{\eta_{j|l}} w^{\alpha_{j|i}}, \mathtt{pk}_{j|i})^{\theta_j}. \end{aligned}$$

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 $(\theta_1, ..., \theta_t)$ are elements of $\mathbb{Z}_p$), 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 auditing 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 $(\mathtt{pk}_1, ..., \mathtt{pk}_d)$, then batch auditing on t auditing tasks can be further optimized as follows

$$e(\prod_{j=1}^{t}\prod_{i=1}^{d}\beta_{j|i}^{\theta_j}, g) \stackrel{?}{=} \prod_{i=1}^{d} e(\prod_{j=1}^{t}(\prod_{l \in L_{j|i}} H(id_{j|l})^{\eta_{j|l}} \cdot w^{\alpha_{j|i}})^{\theta_j}, \mathtt{pk}_i).$$

In this case, the total number of pairing operations during batch auditing can be significantly reduced to

ReKey*. Given private key $\mathtt{sk}_j = \pi_j$, user u_j generates a random $t-1$ degree polynomial $f_j(x)$ as

$$f_j(x) = a_{j,t-1}x^{t-1} + a_{j,t-2}x^{t-2} + \cdots + a_{j,1}x + a_{j,0},$$

where $(a_{j,t-1}, ..., a_{j,1}) \xleftarrow{R} \mathbb{Z}_p^*$ and $a_{j,0} = \pi_j$. Then, user u_j computes s points $(x_1, y_{j,1}), ..., (x_s, y_{j,s})$ based on polynomial $f_j(x)$, where $y_{j,l} = f_j(x_l)$, for $l \in [1, s]$, is a piece of user u_j's private key and $(x_1, ..., x_s)$ are public.
Proxy $\mathtt{P}_l$ generates a piece of a re-signing key as follows: (1) proxy $\mathtt{P}_l$ generates a random $r \in \mathbb{Z}_p^*$ and sends it to user u_i; (2) user u_i computes and sends r/π_i to user u_j, where $\mathtt{sk}_i = \pi_i$; (3) user u_j computes and sends $ry_{j,l}/\pi_i$ to server S_l, where $y_{j,l} = f_j(x_l)$ is a piece of private key $\mathtt{sk}_j = \pi_j$; (4) proxy $\mathtt{P}_l$ recovers $\mathtt{rk}_{i\to j,l} = y_{j,l}/\pi_i \in \mathbb{Z}_p$, where $\mathtt{rk}_{i\to j,l}$ is a piece of re-signing key $\mathtt{rk}_{i\to j}$.

ReSign*. Given public key $\mathtt{pk}_i$, signature σ_k, block m_k and block identifier id_k, the cloud first checks

$$e(\sigma_k, g) \stackrel{?}{=} e(H(id_k)w^{m_k}, \mathtt{pk}_i).$$

If the equation does not hold, the cloud outputs $\perp$; otherwise, each proxy converts this signature σ_k with its own piece of the corresponding re-signing key. Specifically, proxy $\mathtt{P}_l$ converts its part of signature σ_k as

$$\sigma'_{k,l} = \sigma_k^{\mathtt{rk}_{i\to j,l}} = (H(id_k)w^{m_k})^{y_{j,l}}.$$

Finally, as long as t or more proxies are able to convert their parts correctly, the cloud is able to recover signature σ'_k based on the t parts $(\sigma'_{k,1}, ..., \sigma'_{k,t})$ as

$$\sigma'_k = \prod_{l=1}^{t} {\sigma'_{k,l}}^{F_{j,l}(0)}, \tag{4}$$

where $\sigma'_{k,l}$ is computed by proxies $\mathtt{P}_l$, and $F_{j,l}(x)$ is a Lagrange basis polynomial of polynomial $f_j(x)$ and can be pre-computed as

$$F_{j,l}(x) = \prod_{0<h\le t, h\ne l} \frac{x - x_h}{x_l - x_h}, \quad 1 \le l \le t.$$

Similar as in Algorithm **ReSign** in single proxy model, after the re-signing process in the cloud, the original user removes user u_i's id from the user list (UL) and signs the new UL.

Fig. 7. Details of **ReKey*** and **ReSign***.

only $d+1$, which can further improve the efficiency of batch auditing. The correctness of the above equation can be proved as

$$\begin{aligned}
\mathtt{LHS} &= \prod_{j=1}^{t} e(\prod_{i=1}^{d} \beta_{j|i}, g)^{\theta_j} \\
&= \prod_{j=1}^{t}\prod_{i=1}^{d} e(\prod_{l\in L_{j|i}} H(id_{j|l})^{\eta_{j|l}} w^{\alpha_{j|i}}, \mathtt{pk}_i)^{\theta_j} \\
&= \prod_{i=1}^{d} e(\prod_{j=1}^{t}(\prod_{l\in L_{j|i}} H(id_{j|l})^{\eta_{j|l}} \cdot w^{\alpha_{j|i}})^{\theta_j}, \mathtt{pk}_i).
\end{aligned}$$

6.3 Reliability of Panda

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 sometimes 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 ownership 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 Panda*, in the multi-proxy model.

By leveraging an (s,t)-Shamir Secret Sharing ($s \ge 2t-1$) [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 Panda*, 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].

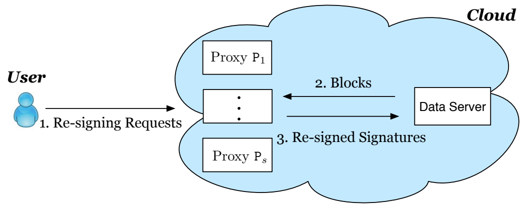


Fig. 8. Multiple re-signing proxies in the cloud.

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 Panda*, most of algorithms are as the same as in Panda, except the two algorithms for generating re-signing keys and re-signing signatures, denoted as **ReKey*** and **ReSign*** respectively. We use $*$ to distinguish them from the corresponding algorithms in the single proxy model. Details of Algorithm **ReKey*** and **ReSign*** are described in Fig. 7.

According to polynomial interpolation, we have $f_j(x) = \sum_{l=1}^{t} y_{j,l} \cdot F_{j,l}(x)$. The correctness of the recovery and transformation of signature σ'_k (i.e., Equation 4) can be explained as

$$\begin{aligned}
\prod_{l=1}^{t} {\sigma'_{k,l}}^{F_{j,l}(0)} &= \prod_{l=1}^{t} (H(id_k)w^{m_k})^{y_{j,l}F_{j,l}(0)} \\
&= (H(id_k)w^{m_k})^{\sum_{l=1}^{t} y_{j,l}F_{j,l}(0)} \\
&= (H(id_k)w^{m_k})^{f_j(0)} \\
&= (H(id_k)w^{m_k})^{\pi_j} = \sigma'_k.
\end{aligned}$$

7 Overhead Analysis

Communication Overhead. According to the description in Section 5, during user revocation, our mechanism does not introduce communication overhead to existing users. 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 $\mathbb{Z}_q$. The size of an auditing proof $\{\boldsymbol{\alpha}, \boldsymbol{\beta}, \{id_l, s_l\}_{l \in L}\}$ 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 $\mathbb{G}_1$ or $\mathbb{Z}_p$, $|id|$ is the size of a block identifier. Therefore, the total communication overhead of an auditing task is $2d \cdot |p| + c \cdot (|id| + |n| + |q|)$ bits.

Computation Overhead. 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 $2\texttt{Exp}_{\mathbb{G}_1} + \texttt{Mul}_{\mathbb{G}_1} + 2\texttt{Pair} + \texttt{Hash}_{\mathbb{G}_1}$, where $\texttt{Exp}_{\mathbb{G}_1}$ denotes one exponentiation in $\mathbb{G}_1$, $\texttt{Mul}_{\mathbb{G}_1}$ denotes one multiplication in $\mathbb{G}_1$, $\texttt{Pair}$ denotes one pairing operation on $e : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_2$, and $\texttt{Hash}_{\mathbb{G}_1}$ denotes one hashing operation in $\mathbb{G}_1$. Moreover, the cloud can further reduce the computation cost of the re-signing on a block to $\texttt{Exp}_{\mathbb{G}_1}$ by directly re-signing it without verification, because the public auditing performed on shared data ensures that the re-signed blocks are correct. Based on Equation (2), the computation cost of an auditing task in our mechanism is $(c+d)\texttt{Exp}_{\mathbb{G}_1} + (c+2d)\texttt{Mul}_{\mathbb{G}_1} + (d+1)\texttt{Pair} + d\texttt{Mul}_{\mathbb{G}_2} + c\texttt{Hash}_{\mathbb{G}_1}$.

8 Performance Evaluation

In this section, we evaluate the performance of our mechanism in experiments. We utilize Pairing Based Cryptography (PBC) Library [27] 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 $\mathbb{G}_1$ or $\mathbb{Z}_p$ is $|p| = 160$ bits, the size of an element of $\mathbb{Z}_q$ 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 Panda 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. In the following experiments, the performance of the straightforward solution is evaluated based on [4] (denoted as SW in this paper), because it is also a public auditing mechanism designed based on bilinear maps.

8.1 Performance of User Revocation

Comparison with the Same Computation Ability. In this experiment, we assume the cloud and an existing user have the same computation ability (Intel Core i5 2.5 GHz Processor and 4 GB Memory) to perform user revocation. We also assume the download speed and upload speed for the data storage and sharing services is 1Mbps and 500Kbps, respectively. Let k denote the number of re-signed blocks during user revocation.

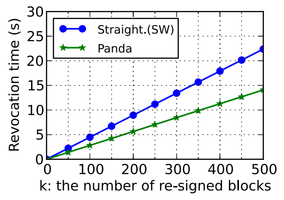


Fig. 9. Impact of k on revocation time (s).

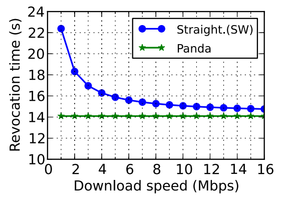


Fig. 10. Impact of the download speed (Mbps) on revocation time (s).

The performance comparison between Panda and the straightforward method during user revocation is presented in Fig. 9. With our mechanism, the cloud is able to not only efficiently re-sign blocks but also save existing users' computation and communication resources. As shown in Fig. 9, when the number of re-signed blocks is 500, which is only 0.05% of the total number of blocks, the cloud in Panda 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. 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 ability in this experiment, it is easy to see that the gap in terms of revocation time between the two lines in Fig. 9 is mainly introduced by downloading the re-signed blocks.

It is clear that if an existing user could have a faster download speed, the gap in terms of revocation time between our mechanism and the straightforward method can be significantly reduced. We can observe from Fig. 10 that, if we assume the number of re-signed blocks is fixed ($k = 500$), the revocation time performed independently by an existing user is approaching to the revocation time of our mechanism when the download speed of data storage and sharing services is increasing. Unfortunately, it is generally impractical for a normal user to maintain or reserve a very high download speed for only a single service on his/her computer. Even if it is possible, with the straightforward method, this user still has to download those 500 re-signed blocks first, which costs him/her additional bandwidth during user revocation compared to our mechanism.

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 according to our experiments. 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 in the straightforward solution. Even if an existing user directly re-signs the blocks without verification, compared to Panda, this user still needs to spend some extra time to download the blocks. As illustrated in Fig. 11, when the number of re-signed blocks is still 500, the cloud in Panda can re-sign these blocks in about 0.14 seconds; while an existing user needs about 8.43 seconds by himself/herself with the straightforward solution.

Similarly, as presented in Fig. 12, the revocation time of Panda is independent with the download speed while an existing user with the straightforward method is able to re-sign the blocks sooner if the download speed is faster. With the comparison between Fig. 9 and Fig. 11, 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. As shown in Fig. 9 and Fig. 11, the key advantage of Panda is that we can improve the efficiency of user revocation and release existing users from the communication and computation burden introduced by user revocation.

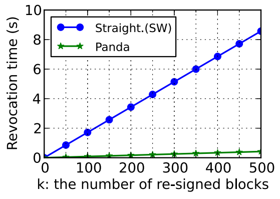


Fig. 11. Impact of k on revocation time (s) without verification.

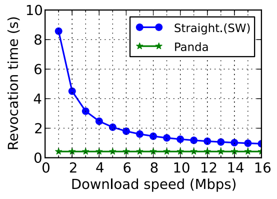


Fig. 12. Impact of of the download speed (Mbps) on revocation time (s) without verification.

Comparison with Different Computation Abilities. In the previous experiment, we assume an existing user and the cloud have the same level of computation ability. Unfortunately, in practical cases, it is common that an existing user may use devices with limited computation resources, such as mobile phones, to access his/her shared data in the cloud. Thus, the computation abilities between the cloud and an existing user are *not symmetric* in data sharing services. In these cases, Panda can save even more revocation time for an existing user than asking this user to re-signing blocks by himself/herself with the straightforward method. To demonstrate the performance advantage of our mechanism in those cases, we use the same parameters as in the previous experiment, except that we assume the existing user is using a resource limited device (e.g., a mobile phone), which has a weaker computation ability than the cloud. Specifically, by leveraging jPBC library [28], we evaluate the performance of the straightforward method on a mobile phone (Motorola MB860, OS: Android 2.3; CPU: Dual-core 1 GHz).

As we can see from Fig. 13 and Fig. 14, the revocation time with the straightforward solution on a mobile phone is dramatically and linearly increasing with the number of re-signed blocks. Specifically, when the number of revoked blocks is 500, the revocation time with the straightforward method on a mobile phone is over 350 seconds, while the revocation time with our mechanism is only less than 4% of it. If both of the cloud and an existing user directly re-sign blocks without verification, our mechanism is also able to save amount of time for an existing user when user revocation happens.

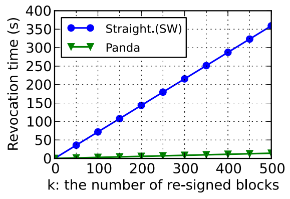


Fig. 13. Impact of k on revocation time (s) on mobile.

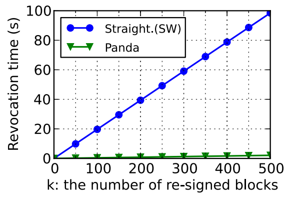


Fig. 14. Impact of k on revocation time (s) without verification on mobile.

Comparison with the Multi-Proxy Model. In Section 6, we proposed to utilize an (s, t)-Shamir Secret Sharing and a number of s multiple proxies to improve the reliability of our mechanism. With Panda* in the multi-proxy model, even if $t-1$ proxies are compromised by adversaries, our mechanism is still able to successfully to convert signatures during user revocation and prevent the arbitrary use of re-signing keys. Clearly, the increase of t will increase the reliability of our mechanism. As a necessary trade-off, the increase of t will inevitably decrease the performance of our mechanism in terms of revocation time. As show in Fig. 15 and Fig. 16, the revocation time of our mechanism in multi-proxy model is linearly increasing with an increase of k and an increase of t. However, these efficiency losses introduced by the multi-proxy model is still acceptable.

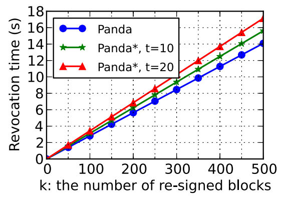


Fig. 15. Impact of k on revocation time (s) in multi-proxy model.

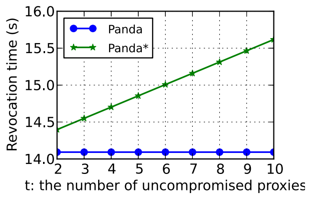


Fig. 16. Impact of t on revocation time (s) in multi-proxy model ($k = 500$).

8.2 Performance of Auditing

Independent Auditing. We can see from Fig. 17 and Fig. 18 that, in order to maintain a higher detection probability for a single independent auditing, a verifier needs more time and communication overhead to finish the auditing task on shared data. Although the auditing time (the time that a public verifier needs to verify the correctness of an auditing proof based on Equation (5)) and communication cost are linearly increasing with the number of existing users in the group, our mechanism is still quite efficient for supporting large groups. Specifically, as we can observed from Fig. 17 and Fig. 18, when the number of existing users is $d = 50$, our mechanism can finish an auditing task with only 14 KB and 860 milliseconds by choosing $c = 460$.

In order to further reduce such kind of increases on computation and communication overhead introduced by a larger number of users, some recent work [29] motivated by our mechanism can be utilized. Particularly, by performing aggregation on signatures generated by different users, the computation and communication overhead in [29] is independent with regard to the number of users in the group. Further details about this aggregation can be found in [29].

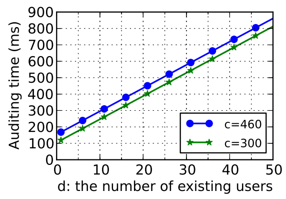


Fig. 17. Impact of d on auditing time (ms).

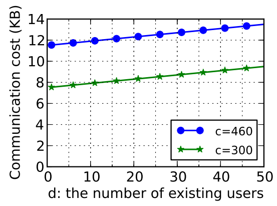


Fig. 18. Impact of d on communication cost (KB).

Batch Auditing. As introduced in Section 6, when a public verifier receives amount of auditing requests in a very short time, our mechanism allows this verifier to perform batch auditing to improve the performance on multiple auditing tasks. We can see from Fig. 19 and Fig. 20 that, with batch auditing, the average auditing time spent on each auditing task can be efficiently reduced. Specifically, according to Fig. 19, if there are 20 auditing tasks, the average auditing time per auditing task with batch auditing is around 270 milliseconds while the average auditing time per task with independent auditing is 295 milliseconds. Moreover, as shown in Fig. 20, if those multiple tasks are all from the same group, then the performance of batch auditing can be significantly improved due to the dramatic decrease on the number of pairing operations, which we discussed in Section 6.

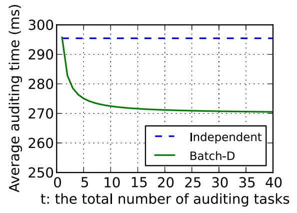


Fig. 19. Impact of t on average auditing time (ms) per task where $d = 10$.

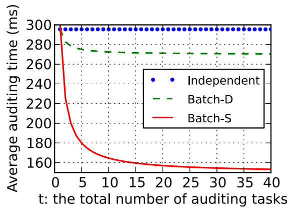


Fig. 20. Impact of t on average auditing time (ms) per task where $d = 10$.

9 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 authenticators and sampling strategies, the verifier is able to publicly audit the integrity of data without retrieving the entire data, which is referred to as public verifiability or public auditing. Shacham and Waters [4] designed an improved PDP scheme based on BLS (Boneh-Lynn-Shacham) signatures [30].

To support dynamic operations on data during auditing, Ateniese *et al.* [31] 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.* [6] utilized the Merkle Hash Tree to support fully dynamic operations in a public auditing mechanism. Erway *et al.* [32] introduced Dynamic Provable Data Possession by using authenticated dictionaries, which are based on rank information. Zhu *et al.* [8] 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.* [5] 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.* [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 computation 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 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 [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.

10 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 Verifiability 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 Dynamic 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.

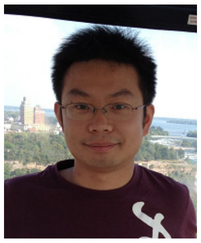
[32] C. Erway, A. Kupcu, C. Papamanthou, and R. Tamassia, "Dynamic Provable Data Possession," in *the Proceedings of ACM CCS 2009*, 2009, pp. 213–222.

[33] B. Chen, R. Curtmola, G. Ateniese, and R. Burns, "Remote Data Checking for Network Coding-based Distributed Stroage Systems," 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.



Boyang Wang is a Ph.D. student from the School of Telecommunications Engineering, Xidian University, Xi'an, China. He was a visiting Ph.D. student at the Department of Electrical and Computer Engineering, University of Toronto, from Sep. 2010 to Aug. 2012. He obtained his B.S. in information security from Xidian University in 2007. His current research interests focus on security and privacy issues in cloud computing, big data, and applied cryptography. He is a student member of IEEE.

Baochun Li is a Professor at the Department of Electrical and Computer Engineering at the University of Toronto, and holds the Bell University Laboratories Endowed Chair in Computer Engineering. His research interests include large-scale multimedia systems, cloud computing, peer-to-peer networks, applications of network coding, and wireless networks. He is a member of ACM and a senior member of IEEE.

Hui Li is a Professor at the School of Telecommunications Engineering, Xidian University, Xi'an, China. He received B.Sc. degree from Fudan University in 1990, M.Sc. and Ph.D. degrees from Xidian University in 1993 and 1998. In 2009, he was with Department of ECE, University of Waterloo as a visiting scholar. His research interests are in the areas of cryptography, security of cloud computing, wireless network security, information theory. He is the co-author of two books. He served as TPC co-chair of ISPEC 2009 and IAS 2009, general co-chair of E-Forensic 2010, ProvSec 2011 and ISC 2011.