CoRPA: A Novel Efficient Shared Data Auditing Protocol in Cloud Storage

Reyhaneh Rabaninejad, Mahmoud Ahmadian Attari, Maryam Rajabzadeh Asaar, Mohammad Reza Aref

Abstract—As data sharing has become one of the most popular features offered by cloud storage services, designing public auditing mechanisms for integrity of shared data stored at the cloud becomes much more important. Two unique problems which arise in shared data auditing mechanisms include preserving signer identity privacy and collusion resistant revocation of cloud users. When the data stored at the cloud is shared among a group of users, different users may modify different data blocks; therefore, data blocks are signed by different users which accordingly leak signer identities to the public verifier. Also, when a user is revoked from the group, the signatures generated by this user become invalid to the group and should be re-signed by the cloud server using re-signature keys. In addition, the collusion of cloud server who possess re-signature keys and the revoked user should leak no information about the private key of other users. In this paper, by employing a collusion resistant proxy re-signature scheme, we propose a public auditing mechanism for integrity of shared data that provides identity privacy and collusion resistant user revocation, simultaneously. We also formally prove the mentioned properties based on exact security definition and well-known hard problems in the random oracle model. To our best knowledge, this paper presents the first public auditing mechanism based on collusion resistant proxy re-signatures. Moreover, our protocol supports large dynamic group of users, batch verification of multiple auditing tasks and fully dynamic data operations, efficiently. Overhead analysis and experimental results demonstrate the excellent efficiency of our scheme in comparison to the state of the art.

Index Terms—Cloud storage, shared data, public auditing, identity privacy, user revocation, collusion resistancy.

1 Introduction

Cloud storage, as one of the most important services provided by cloud computing, has caused a great migration of data from local storage systems to the cloud as it offers efficient and scalable services with a lower cost. Since the data owner, has no physical control over the data after outsourcing, the data integrity is subject to some risks such as being lost or corrupted [1]. To audit the integrity of the outsourced data, Ateniese et al. proposed the notion of provable data possession (PDP) [2]. In PDP, for auditing data integrity, the data is divided into some blocks, a signature is attached to each data block and the blocks together with their signatures are outsourced to the cloud server. The public verifier then can efficiently check data integrity without the need to download the entire data, by challenging a small set of randomly chosen data blocks and verifying the proof that the server returns. This public verifier can be a client as the data user, or a third party auditor (TPA) who has the resources to provide data verification services to users. Due to its necessity, PDP has attracted extensive research interest (e.g. [2]–[7]) in recent years.

Beside storage, data owners often use the cloud storage applications, like Dropbox and Google Drive, to *share* their data with other cloud users. Wang et al. proposed a protocol called Oruta for auditing the data shared by a group of users [8]. A problem which arises during shared data auditing in the cloud, is how to preserve identity of the signers from the TPA. Because using the signer identities, the TPA can discover which user in the group or block in shared data has more importance than others. For example, when a user in the group signs more blocks, then probably that user has more important role in the group; also, when a block in shared data is frequently updated by different users, this block probably contains high value data. Therefore, *identity privacy* is an important issue in shared data auditing which keeps such sensitive information private from the TPA. To provide the identity privacy, Wang et al. [8] employed ring signatures [9] to generate metadata on each block. But due to the use of ring signatures, Oruta only supports *static* groups and membership of users cannot change during data sharing. Since the signature length on each block is equal to the number of group users, to add a new user into the group or revoke an existing user, the group manager should download all the outsourced data blocks, re-compute signatures based on the new group size and upload the new signatures to the server. Also, the large signature size and auditing cost in this mechanism, makes it inefficient for practical scenarios.

Later, Wang et al. proposed Panda [10], which is a protocol for shared data auditing that enjoys efficient user revocation. When the data is shared among a group of users, each user is able to modify some data blocks and send each modified block together with its signature to the cloud. When a user leaves the group or misbehaves, he should be revoked from the group and the signatures generated by this user become invalid to the group. Therefore, these signatures should be re-signed so that the data integrity auditing can be performed with the public keys of existing users only. Wang et al. in [10] employed proxy re-signatures [11] which enables the *cloud server* to efficiently re-sign the blocks signed by the revoked user via re-signature keys. The re-signature key $rk_{A\to B}$ in [10], which converts a signature from user A to a signature from user B is generated in a four-step procedure as follows: (1) The server generates a random

- *R. Rabaninejad and M. Ahmadian Attari are with the Department of Electrical Engineering, K. N. Toosi University of Technology, Tehran. Iran. E-mail: rabaninejad@ee.kntu.ac.ir, mahmoud@eetd.kntu.ac.ir.*
- *M. Rajabzadeh Asaar is with the Department of Electrical and Computer Engineering, Science and Research Branch, Islamic Azad University, Tehran, Iran. Email: asaar@srbiau.ac.ir.*
- *M. R. Aref is with the Department of Electrical Engineering, Sharif University of Technology, Tehran. Iran. E-mail: aref@sharif.edu.*

- *This work was partially supported by Iran NSF under Grant No. 92.32575*

number $r \in \mathbb{Z}_q$ and sends it to user A; (2) User A sends r/a to user B, where a is the private key of A; (3) User B sends rb/a to the server, where b is the private key of B; (4) The server recovers $rk_{A \to B} = b/a$. However, it is assumed that private channels exist between each pair of entities in each step of the above procedure and also there is no collusion between the server and any user. If the cloud server (who possess $rk_{A \to B} = b/a$) and a revoked user (e.g. user B with private key b) collude, they can easily recover the private key of user A. Also, Panda does not provide data and identity privacy which was a distinctive feature of Oruta. Furthermore, the auditing cost of the scheme is linear to the number of users which makes it inefficient for large groups. Yuan and Yu proposed another shared data auditing scheme with user revocation utilizing polynomial-based authentication tags [12]. To revoke a user, the group manager generates two group elements as randomized pieces of the re-signature key and sends one element to the cloud and the other to the TPA. Using these two randomized pieces of the re-signature key, the cloud and the TPA together are able to correctly re-sign the blocks signed by the revoked user [12]. Since the server does not possess the exact re-signature keys, the scheme is collusion-resistant. However, Yu et al. in [13] showed that the scheme in [12] is vulnerable to the collusion of the server and revoked user. Specifically, if the server collude with a revoked user, they can deceive the TPA and pass the verification although the data has been corrupted. Furthermore, since the TPA is involved in the user revocation, the scheme in [12] has more communication and computation costs during user revocation in comparison to Panda. Also, the same as Panda, it does not provide identity privacy. Jiang et al. [14] also considered the problem of secure user revocation where by employing the group signatures [15], they prevent the collusion of cloud and revoked users. However, the expensive computation cost of generating group signatures and also costly auditing operations, make the scheme inefficient. Furthermore, no formal proof is provided in the paper to show that the scheme is collusion resistant [14].

In this paper, we propose a **Co**llusion **R**esistant **P**ublic **A**uditing (**CoRPA**) protocol, for integrity of shared data that simultaneously provides identity privacy and collusion resistant user revocation, as the most significant features of shared data auditing. To this end, we first propose a collusion resistant homomorphic authenticable proxy re-signature scheme based on the scheme in [16] and prove its unforgeability and collusion resistancy. Next, we employ the proposed proxy re-signature scheme for constructing homomorphic tags in CoRPA. In our mechanism, tag (signature) on each data block is generated in two phase: first, the user generates a first-level signature on the data block and sends the (block,signature) pair to the cloud server. Second, the server, using the re-signature keys, translates the received signature from the user to a second-level signature from the system on the same block and saves (block,translated-signature) pair in his storage. This signature translation done by server has three advantages. First, the user only computes a linear combination as the first-level signature and offloads the exponentiation part of tag generation to the server which improves the efficiency of tag generation at the user side. Second, it preserves identity privacy of the group users: since the translated-signature is a system signature, no one can distinguish who the signer of a special block is. Third, it enables efficient user revocation: when a user is revoked from the group, since all signatures are previously translated to the system signature, the re-signing of the blocks signed by the revoked user is not needed. Furthermore, during the generation of re-signature keys in CoRPA, no private channel between entities is assumed as oppose to Panda [10]. Meanwhile, the server who possess the re-signature keys, is not able to independently generate a valid signature on an arbitrary block on behalf of users or system even if he collude with one of the group users (e.g. a revoked user), which will be formally proved in the paper.

Moreover, we mention that the proposed scheme is scalable. First, in CoRPA, signature generation cost at the user side and also computation and communication costs at the server and verifier sides are constant and do not change with the number of group users, no matter how many users access the data blocks. Accordingly, our mechanism supports data sharing among a large dynamic group of users efficiently. Second, our scheme enables the TPA to handle multiple auditing tasks simultaneously and efficiently via batch auditing. Furthermore, by utilizing index hash tables [17], all users who access the shared data, can efficiently modify data blocks including insert a new block, and delete or update an existing block at any time. Table 1, includes a high-level comparison between CoRPA and selected previous protocols.

Our Contributions. Our contribution can be summarized as follows:

1) We design a public auditing protocol for integrity of shared data which is based on a new collusion resistant proxy re-signature scheme proposed in this paper. To our best knowledge, this paper presents the first public auditing mechanism based on collusion resistant proxy re-signatures.
2) Our public auditing protocol is fully privacy-preserving. That is, no information of the data content or signer identities is leaked to the TPA during the auditing protocol. At the same time, our mechanism provides collusion resistant revocation of cloud users, as the most significant feature of shared data auditing. We also formally prove the privacy-preserving and collusion resistancy properties of our protocol in the random oracle model.
3) In our protocol, as oppose to [8] and [10], signature generation and server and verifier computation and communication costs are not dependent on the number of group users; so, it supports large dynamic group of users efficiently. Batch verification of multiple auditing tasks and fully dynamic data operations are also provided in our scheme. Moreover, overhead analysis and experimental results demonstrate the excellent efficiency of our scheme in comparison to the state of the art.

The paper is organized as follows. In Section 2, we present the system model, security model and design goals. Section 3, gives background on some cryptographic primitives used in CoRPA. In Section 4, we propose a collusion resistant homomorphic authenticable proxy re-signature scheme which is used as a building block in CoRPA. Detailed design and security analysis of our mechanism are proposed in Section 5. Section 6 gives the performance evaluation, Section 7 briefly discusses the related work and finally Section 8 concludes the paper.

TABLE 1
Feature list and comparison between CoRPA and some existing well-known schemes

Metric \ Scheme	Oruta [8]	Panda [10]	CoRPA
Data Dynamics	Yes	Yes	Yes
Identity and Data Privacy	Yes	No	Yes
Efficient User Revocation	No	Yes	Yes
Collusion Resistancy	–	No	Yes

2 Problem Statement

2.1 System and Security Model

As described in Fig. 1., the system model in this paper includes four entities: key generation center (KGC), the cloud server, a group of users U, and the third party auditor (TPA). The trusted KGC, generates a system public/private key pair (pk_S, sk_S) and re-signature key $rk_{i\to S}$ for all $i \in U$ which converts a signature from user i to a signature from the system. All the re-signature keys are published publicly. Also, the system private key is only held by the KGC.

A user who is the data owner, originally creates the data which is divided into a number of blocks and shares it with a group of users through the cloud. This group can dynamically change during data sharing: adding new members to the group or revoking the existing ones can be easily done with a minimum overhead. All the group members (including the data owner) are able to access the shared data and modify it. Modification includes insert, delete or update operations on the blocks.

When the data is created for the first time, and also each time a data block is modified, the respective user signs the new/modified block and sends the (block,signature) pair to the cloud server. The server, using the re-signature keys gained from the KGC, translates the received signature from the user to a signature from the system on the same block. Then the server saves (block,translated-signature) pair in his storage. This signature translation by server has three advantages. First, the user only computes a linear combination as the first-level signature and offloads the exponentiation part of tag generation to the server which improves the efficiency of tag generation. Second, it preserves identity privacy of the group users: since the translated-signature is a system signature, no one can distinguish who the signer of a special block is. Third, it enables efficient user revocation: when a user is revoked from the group, all the blocks signed by the revoked user should be re-signed, but here since all signatures are previously translated to the system signature, this re-signing operation is not needed. So, to revoke a user id, the data owner as the group manager just sends the revoked id to the server to update his revocation list (RL).

To audit the integrity of shared data, the TPA sends an auditing challenge to the cloud server. Based on the challenged blocks and their signatures, the cloud server generates a proof and sends it as a response to the TPA. Finally, the TPA verifies the received proof which demonstrates the data correctness.

There are two security issues which should be considered in a public auditing mechanism:

Security Against Server. In public auditing mechanisms, the cloud server is not fully trusted in the sense that it may hide data loss/corruption from verifiers in order to save its reputation. Also, the server who possess the re-signature keys, should not be able to independently generate a valid signature on an arbitrary block on behalf of users or system even if he *collude* with one of the group users (e.g. a revoked user).

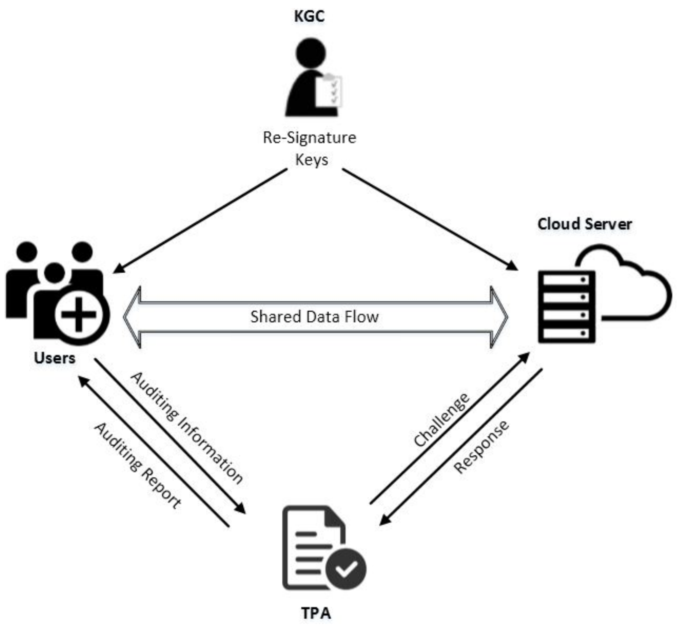

Fig. 1. The system model in this paper

Privacy Against TPA. The TPA who provides data integrity verification services to users, is not allowed to learn anything about both the data content and signer identities which are confidential to the group users. During the auditing protocol, some information is given to the TPA so that it is able to verify the data integrity. The TPA, using the gained information, may try to reveal the data content or signer identities and violate the users' privacy.

2.2 Design Goals

Our design goals can be listed as follows:

1) *Public Auditing:* To allow anyone (not just the data users) to audit shared data integrity without the need to download the data from the cloud.
2) *Unforgeability and Collusion Resistancy:* Only a user in the group with a proper private key can generate a valid tag (signature) on the shared data blocks. Also, collusion of the server who possess re-signature keys and one of the delegation partners, reveals no information about the private key of another partner.
3) *Correctness:* A public verifier is able to correctly audit the integrity of shared data.

4) *Soundness:* The untrusted cloud server cannot forge a valid proof in the auditing process.
5) *Zero-knowledge Identity and Data Privacy:* To ensure that the identity of the signer on each block in shared data and also the data content is not revealed to the public verifier during the auditing process.
6) *Efficient and Secure User Revocation:* To enable efficient revoking of group users. Also, to ensure that the revoked users can no longer generate valid signatures on shared data.
7) *Scalability:* To enable efficient data sharing among a large dynamic group of users. Also, to allow the public verifier to handle multiple auditing tasks simultaneously and efficiently.
8) *Dynamic Data Operations:* To allow all users who access the shared data to efficiently modify data blocks at any time.
9) *Lightweight:* To enable low overhead tag generation for the users and low computation and communication overhead for the public verifier.

3 Preliminaries

In this section we review bilinear maps, homomorphic authenticators, and proxy re-signatures as the preliminaries used in our design. We also introduce security assumptions which are employed to prove our protocol's security.

3.1 Bilinear Maps

Let G_1 and G_2 be two multiplicative cyclic groups of prime order q, g be a generator of G_1 and e be a bilinear map where $e : G_1 \times G_1 \to G_2$. The bilinear map e is a function with the following properties:

1) *Bilinearity*: For all $u, v \in G_1$ and $a, b \in Z_q$, $e(u^a, v^b) = e(u, v)^{ab}$.
2) *Non-degeneracy*: $e(g, g) \neq 1$, where 1 denotes the identity element of G_2.
3) *Computability*: There is an efficient algorithm to compute $e(u, v)$ for $u, v \in G_1$.

3.2 Homomorphic Authenticators

Homomorphic authenticators or homomorphic verifiable tags are verification metadata generated for individual data blocks which are used as a building block in the public auditing schemes [2], [3], [18]. Consider a signer with public/private key pair (pk, sk) and two data blocks $m_1, m_2 \in \mathbb{Z}_q$ with signatures σ_1, σ_2, respectively. Homomorphic authenticators have three following properties:

1) *Unforgeability*: A valid signature can only be generated by a user who possess a private key.
2) *Non-malleabilty*: Given m_1 and m_2, and their signatures σ_1 and σ_2, a user who does not have the private key sk, is not able to generate a valid signature on a block $m' = \alpha_1 m_1 + \alpha_2 m_2 \in \mathbb{Z}_q$ by combining σ_1 and σ_2, where α_1 and α_2 are random values in $\mathbb{Z}_q$.
3) *Blockless verifiability*: Given a block $m' = \alpha_1 m_1 + \alpha_2 m_2 \in \mathbb{Z}_q$, where α_1 and α_2 are random values in $\mathbb{Z}_q$, and given σ_1 and σ_2, a verifier can check the correctness of m' without knowing m_1 and m_2.

The first two properties (unforgeability and non-malleabilty) guarantees tag security. That is, no one can generate a valid signature on a new block unless he has a proper private key. Blockless verifiability allows a public verifier to audit the cloud data integrity with only having a linear combination of the challenged blocks and without the need to download the entire data which is not efficient and secure. The integrity of the linear combination is verified iff all the individual data blocks are intact.

3.3 Proxy Re-Signatures

In a proxy re-signature scheme, the proxy is given a re-signature key $rk_{A \to B}$, to *translate* a valid signature from user A on message m, $\sigma_A(m)$, into a valid signature from user B on the same message, $\sigma_B(m)$. In other words, B delegates to A, via a proxy, the ability to change A's signatures into B's. So, three parties are involved in the scheme: B as the delegator, A as the delegatee and proxy as the translator. This primitive was first proposed by Blaze et al. [11] and later followed up by Ateniese et al. in [16] who proposed an exact security model by considering two types of adversaries: outside adversary (external security) and inside adversary (internal security) which are defined in the following:

External Security. This notion considers security against adversaries outside the system which is equivalent to *strong* existential unforgeability under adaptive chosen-message attack (the adversary cannot create a new signature even for a previously signed message).

Internal Security. Internal adversaries include the proxy, the delegator and the delegatee who are inside the system. Here, three types of security can be considered:

1. Security Against the Proxy: If the delegator and the delegatee are both honest, the proxy cannot generate a valid signature on arbitrary message on behalf of the delegator or the delegatee. The only operation the proxy can do is to translate a signature generated by one of the delegatees to a signature from the delegator.

2. Delegator Security: The honest delegator is secure against a colluding delegatee and proxy. In other words, the delegatee and the proxy cannot produce any *first-level* signatures on the delegator's behalf, even if they collude. Note that a second-level signature can be easily generated by the colluding delegatee and proxy using the re-signature key. The concept of first and second level signatures will be introduced in Section 4.

3. Delegatee Security: The honest delegatee is secure against a colluding delegator and proxy. In other words, the delegator and the proxy cannot produce any signatures on the delegatee's behalf, even if they collude.

3.4 Security Assumptions

The security of the proposed mechanism in this paper is based on Square Diffie-Helman, Discrete Logarithm and 2-Discrete Logarithm assumptions which are described in the following:

Definition 1: Square Diffie-Helman (sq-DH) Assumption. For each PPT adversary $\mathcal{A}$ which is given (g, g^a), $\mathcal{A}$'s advantage to compute g^{a^2} is negligible, where $g \in G_1$ and $a \in \mathbb{Z}_q$ are randomly chosen from the corresponding groups. In other words, for any PPT algorithm $\mathcal{A}$, we have:

$$Pr[\mathcal{A}_{sq-DH}(g, g^a) = (g^{a^2}) : a \overset{R}{\leftarrow} \mathbb{Z}_q] \leq \epsilon \qquad (1)$$

where ϵ is a negligible function. It can be easily shown that the sq-DH problem is equivalent to the computational Diffie-Helman (CDH) problem [16].

Definition 2: Discrete Logarithm (DL) Assumption. For each PPT adversary $\mathcal{A}$ which is given (g, g^a), $\mathcal{A}$'s advantage to compute a is negligible, where $g \in G_1$ and $a \in \mathbb{Z}_q$ are randomly chosen from the corresponding groups. In other words, for any PPT algorithm $\mathcal{A}$, we have:

$$Pr[\mathcal{A}_{DL}(g, g^a) = (a) : a \overset{R}{\leftarrow} \mathbb{Z}_q] \leq \epsilon \quad (2)$$

where ϵ is a negligible function.

Definition 3: 2-Discrete Logarithm (2-DL) Assumption. For each PPT adversary $\mathcal{A}$ which is given (g, g^a, g^{a^2}), $\mathcal{A}$'s advantage to compute a is negligible, where $g \in G_1$ and $a \in \mathbb{Z}_q$ are randomly chosen from the corresponding groups. In other words, for any PPT algorithm $\mathcal{A}$, we have:

$$Pr[\mathcal{A}_{2-DL}(g, g^a, g^{a^2}) = (a) : a \overset{R}{\leftarrow} \mathbb{Z}_q] \leq \epsilon \quad (3)$$

where ϵ is a negligible function.

4 A New Homomorphic Authenticable Proxy Re-Signature Scheme

In this section, we propose a new *collusion resistant* homomorphic authenticable proxy re-signature scheme named as HAS_{uni} which is based on the S_{uni} signature proposed by Ateniese et al. in [16]. Next, in Section 5, we employ HAS_{uni} to build our collusion resistant public auditing protocol.

4.1 Construction of HAS_{uni}

The scheme consists of six algorithms **Setup**, **KeyGen**, **ReKey**, **Sign**, **ReSign**, **Verify** which are described in Fig. 2. Correctness of the verification equation in **Verify** can be easily shown as: $e(g, s) = e(g, h^{a(m+e+H(id\|r))}) = e(pk^{(1)}, rh^{m+H(id\|r)})$. The properties of HAS_{uni} can be listed as below:

- **Homomorphic Authenticable:** S_{uni} proposed in [16] is not blockless verifiable, which means that in the public auditing protocol, the public verifier should download the entire data to check data integrity which is not efficient and secure. HAS_{uni} (Homomorphic Authenticable S_{uni}), can be efficiently used in the auditing protocol without the need to download the data.
- **Collusion Resistant:** As far as we know, HAS_{uni} is the first collusion resistant homomorphic authenticable proxy re-signature scheme. Previous homomorphic authenticable proxy re-signature schemes such as the one in [10], assume a private channel between each pair of entities in **ReKey** and also assume no collusion happens between them. If the proxy who has $rk_{A\to B}$ and user A or B collude, they can obtain the other user's private key. But in HAS_{uni}, collusion of the proxy and any of users reveals no information which will be shown in the next subsection.
- **Single-Use Unidirectional:** Single-use means that the re-sign function converts a first-level signature to a second-level one, but no other conversion is possible on a second-level signature. Unidirectional means that in HAS_{uni}, as in S_{uni}, the re-signature key $rk_{A\to B}$ can only be used to convert A's signature to B's, but not vice versa.

4.2 Security Analysis of HAS_{uni}

Theorem 1. *HAS_{uni} is secure (unforgeable and collusion resistant) under sq-DH and 2-DL assumptions in the random oracle model.*

Proof. As was noted in Subsection 3.3, in a proxy re-signature scheme two types of adversaries are considered: internal adversaries and external ones. Since in our scheme, the re-signature keys are public, all attackers can act as the proxy and therefore, all are internal. So, it suffice to prove security against inside attackers which include the proxy, the delegator and the delegatee. In our scheme, the server acts as the proxy, the key generation center (KGC) as the delegator and the group users who share a data file as the delegatees. Since the KGC is a fully trusted entity, the third item in the internal security definition in Subsection 3.3 holds trivially. Therefore, only the proxy and delegatees are considered as the internal adversaries and the scheme is proven to be secure against them.

–Security Against the Proxy: A rogue proxy cannot forge a signature on behalf of honest delegator and delegatees using the re-signature keys. To show this, we prove that if a proxy $\mathcal{A}$ can forge a signature with probability ϵ, then there exists an algorithm $\mathcal{B}$ that solves the sq-DH problem in G_1 with probability ϵ^2. In the following game, the adversary $\mathcal{A}$ is given the users' public keys, also is given access to the hash oracle H, the signature oracle $\mathcal{O}_{sign}$ and the re-signature key generation oracle $\mathcal{O}_{rekey}$. $\mathcal{A}$ wins the game if it outputs a valid second-level signature from user t on a pair of block/identifier (m, id) which the tuple (t, m, id) has not never been queried before from the signature oracle.

Setup. Given a sq-DH challenge (g, g^a), $\mathcal{B}$ gives to $\mathcal{A}$ the system parameters as g and $h = g^a$.

Public Keys. To generate the public key of user i, $\mathcal{B}$ chooses a random value $y_i \in \mathbb{Z}_q$ and sets $pk_i = ((g^a)^{y_i}, g^{1/y_i}) = (g^{ay_i}, h^{1/ay_i})$, where the virtual private key is equal to ay_i. Then $\mathcal{B}$ sends pk_i to $\mathcal{A}$ and saves the pair (i, y_i) in a table.

Hash Query. To answer H query on input a_i, $\mathcal{B}$ first checks previously queried values in table T_H. If there is the same entry in T_H, he outputs the corresponding value. Otherwise, he outputs a random value $c_i \in \mathbb{Z}_q$ and saves (a_i, c_i) in T_H.

Signing Query. $\mathcal{A}$ sends a query to $\mathcal{O}_{sign}$ on input (j, m_i, id_i) which means a signature on block/identifier (m_i, id_i) from user j. To answer this query, since $\mathcal{B}$ does not possess sk_j, he generates $s = sk_j(m_i + e + H(id_i \parallel r))(mod q)$ by controlling the output of H as follows. He selects random $s \in \mathbb{Z}_q$ as the queried signature. Also, selects random $c \in \mathbb{Z}_q$ as the output of $H(id_i \parallel r)$ and computes $r = (pk_j^{(2)})^s / h^{m_i+c}$ where $pk_j^{(2)}$ is the second term in user j's public key $pk_j = (pk_j^{(1)}, pk_j^{(2)})$. Finally, $\mathcal{B}$ checks if $(id_i \parallel r, .)$ is an entry in table T_H; if it is so, the random c assigned to $H(id_i \parallel r)$ is not accurate and the game aborts. Otherwise, $\mathcal{B}$ saves $(id_i \parallel r, c)$ in table T_H and outputs the queried first-level signature as $\sigma = (s, r)$.

Re-key Query. A query to $\mathcal{O}_{rekey}$ on input (i, j) means the re-signature key $rk_{i\to j}$. $\mathcal{B}$ generates $rk_{i\to j}$ without the need to any private keys by using the values g^a, y_i and y_j due to the following equalities:

Setup. Consider the map $e : G_1 \times G_1 \rightarrow G_2$ with G_1 and G_2 as groups of prime order q, and g,h as generators of G_1. The public parameters are $(e, q, G_1, G_2, g, h, H)$, where H is a hash function from arbitrary string to elements in $\mathbb{Z}_q$.
KeyGen. User A, on input the global parameters $(e, q, G_1, G_2, g, h, H)$, selects a random $a \in_r \mathbb{Z}_q$ and generates the public key as $pk_A = (g^a, h^{1/a})$ and private key as $sk_A = (a, h^a)$.
ReKey. User B, on input user A's public key pk_A and his private key sk_B, generates the re-signature key $rk_{A\to B}$ as $rk_{A\to B} = (pk_A)^{sk_B} = (h^{1/a})^b = h^{b/a}$. Then B publishes $rk_{A\to B}$ publicly.
Sign. To sign a block m with block identifier id, user A with $sk_A = a$, selects $e \in_r \mathbb{Z}_q$ and outputs $\sigma = (s, r)$ as follows: $r = h^e$ and $s = a(m+e+H(id \parallel r))(modq)$. Here s is called a *first-level* signature.
ReSign. Given the re-signature key $rk_{A\to B}$, the public key pk_A, a first-level signature σ on a block m with block identifier id, the proxy first checks that Verify(pk_A, m, id, σ)= 1. If the verification does not hold, output $\perp$; otherwise output $\sigma' = (s', r')$, where $r' = r$ and $s' = (rk_{A\to B})^s = (h^{b/a})^{a(m+e+H(id\|r))} = h^{b(m+e+H(id\|r))}(modq)$. Here s', which has an exponential form, is called a *second-level* signature.
verify. On input a public key $pk = (pk^{(1)}, pk^{(2)})$, a signature $\sigma = (s, r)$, a block m with block identifier id, check the equation $e(g, s) \overset{?}{=} e(pk^{(1)}, rh^{m+H(id\|r)})$ (If σ is a first-level signature, put $s = h^s$). Output 1 if the equation holds and 0 otherwise.

Fig. 2. Details of HAS_{uni}

$rk_{i\to j} = h^{(sk_j)/(sk_i)} = h^{(ay_j)/(ay_i)} = (g^a)^{y_j/y_i}$.

Forgery. After query phase, $\mathcal{A}$ outputs a second-level signature from user t on a pair of block/identifier (m, id). $\mathcal{A}$ will win the game if the signature is valid and the tuple (t, m, id) has not never been queried before from the signature oracle.

The probability of aborting in the above game, is equal to the probability of collision in H which is at most $q_H/2^k$ where q_H is the total number of queries to H. So, the probability that $\mathcal{A}$ wins the game is $\epsilon(1 - q_H/2^k)$. Due to the Reset Lemma [19], $\mathcal{A}$ can generate two valid signatures (r, c_1, s_1) and (r, c_2, s_2) on the same pair of block/identifier (m, id) for user t, where (c_1, c_2) are two different responses from H on input $(id \parallel r)$ in two different execution of the above game. Using this, $\mathcal{B}$ can solve sq-DH problem by computing:

$$(\frac{s_1}{s_2})^{1/(y_t(c_1-c_2))} = (\frac{h^{ay_t(m+e+c_1)}}{h^{ay_t(m+e+c_2)}})^{1/(y_t(c_1-c_2))} \quad (4)$$
$$= h^{\frac{ay_t(m+e+c_1-m-e-c_2)}{y_t(c_1-c_2)}} = h^a = g^{a^2}.$$

–Delegator Security: The honest delegator is secure against a colluding delegatee and proxy. In other words, the delegatee and the proxy cannot produce any *first-level* signatures on behalf of the delegator, even if they collude (a second-level signature can be easily generated by the colluding delegatee and proxy using the re-signature keys). To show this, we prove that if adversary $\mathcal{A}$ (colluding delegatee and proxy) can forge a signature on the delegator's behalf (here we show the delegator with S) , then there exists an algorithm $\mathcal{B}$ that solves the 2-DL problem in G_1. In the following game, the adversary $\mathcal{A}$ is given all the delegatee's public/private key pairs, the delegator's public key (pk_S), also is given access to the hash oracle H, the signature oracle $\mathcal{O}_{sign}$ and the re-signature key generation oracle $\mathcal{O}_{rekey}$. $\mathcal{A}$ will win the game if it outputs a valid first-level signature from delegator S on a pair of block/identifier (m, id) which has not never been queried before from the signature oracle.

Setup. Given a 2-DL challenge (g, g^a, g^{a^2}), $\mathcal{B}$ gives to $\mathcal{A}$ the system parameters as g and $h = (g^a)^x$, where $x \in_r \mathbb{Z}_q$.

Public and Private Keys. $\mathcal{B}$ sets the delegator public key as $pk_S = (g^a, h^{1/a} = g^x)$. Also, the public/private key pair of user i (delegatee) is set as $pk_i = (g^{y_i}, h^{1/y_i} = g^{ax/y_i})$ and $sk_i = y_i$, where $y_i \in_r \mathbb{Z}_q$.

Hash Query. To answer H query on input a_i, $\mathcal{B}$ first checks previously queried values in table T_H. If there is the same entry in T_H, he outputs the corresponding value. Otherwise, he outputs a random value $c_i \in \mathbb{Z}_q$ and saves (a_i, c_i) in T_H.

Signing Query. $\mathcal{A}$ sends a query to $\mathcal{O}_{sign}$ on input (S, m_i, id_i) which means a signature on block/identifier (m_i, id_i) from delegator S. To answer this query, since $\mathcal{B}$ does not possess sk_S, he generates $s = sk_S(m_i + e + H(id_i \parallel r))(modq)$ by controlling the output of H as follows. He selects random $s \in \mathbb{Z}_q$ as the queried signature. Also, selects random $c \in \mathbb{Z}_q$ as the output of $H(id_i \parallel r)$ and computes $r = (pk_S^{(2)})^s/h^{m_i+c}$ where $pk_S^{(2)}$ is the second term in S's public key $pk_S = (pk_S^{(1)}, pk_S^{(2)})$. Finally, $\mathcal{B}$ checks if $(id_i \parallel r, .)$ is an entry in table T_H; if it is so, the random c assigned to $H(id_i \parallel r)$ is not accurate and the game aborts. Otherwise, $\mathcal{B}$ saves $(id_i \parallel r, c)$ in table T_H and outputs the queried first-level signature as $\sigma = (s, r)$.

Re-key Query. A query to $\mathcal{O}_{rekey}$ on input (i, S) means the re-signature key $rk_{i\to S}$. $\mathcal{B}$ generates $rk_{i\to S}$ as $rk_{i\to S} = (h^{1/y_i})^a = (g^{a^2})^{x/y_i}$.

Forgery. After query phase, $\mathcal{A}$ outputs a first-level signature from delegator S on a pair of block/identifier (m, id). $\mathcal{A}$ will win the game if the signature is valid and the pair (m, id) has not never been queried before from the signature oracle.

As before, the probability that $\mathcal{A}$ wins the game is $\epsilon(1 - q_H/2^k)$ and by applying the Reset Lemma, $\mathcal{B}$ can solve 2-DL problem from the two first-level signatures (r, c_1, s_1) and (r, c_2, s_2) as below:

$$\frac{s_1 - s_2}{c_1 - c_2} = \frac{a(m + e + c_1) - a(m + e + c_2)}{c_1 - c_2} = a. \quad (5)$$

In this theorem, we proved that the rogue entities (the proxy and the delegatees), cannot forge a valid signature on behalf of the delegator or other delegatees even if they collude and therefore HAS_{uni} is secure. □

Theorem 2. *HAS_{uni} is a homomorphic authenticable (blockless verifiable and non-malleable) proxy re-signature scheme.*

Proof. These two properties of HAS_{uni} are proved in the following paragraphs:

–blockless verifiability. The TPA can check the correctness of the block $m' = \alpha_1 m_1 + \alpha_2 m_2$ without knowing m_1 and m_2, if he is given $\alpha_1, \alpha_2 \in_r \mathbb{Z}_q$, and given $\sigma_1 = (s_1, r_1)$ and $\sigma_2 = (s_2, r_2)$ signed by user t with public key pk_t and also given block identifiers id_1, id_2. More precisely, auditing of block m' is done using equation below:

$$e(g, s_1^{\alpha_1} s_2^{\alpha_2}) \overset{?}{=} e(pk_t^{(1)}, r_1^{\alpha_1} r_2^{\alpha_2} h^{m' + \alpha_1 H(id_1 \| r_1) + \alpha_2 H(id_2 \| r_2)}).$$

The correctness of equation above can be simply shown as below:

$$\begin{aligned} & e(g, s_1^{\alpha_1} s_2^{\alpha_2}) = \\ & e(g, h^{\alpha_1 sk_t (m_1 + e_1 + H(id_1 \| r_1))} h^{\alpha_2 sk_t (m_2 + e_2 + H(id_2 \| r_2))}) = \\ & e(g^{sk_t}, r_1^{\alpha_1} r_2^{\alpha_2} h^{\alpha_1 (m_1 + H(id_1 \| r_1)) + \alpha_2 (m_2 + H(id_2 \| r_2))}) = \\ & e(pk_t^{(1)}, r_1^{\alpha_1} r_2^{\alpha_2} h^{m' + \alpha_1 H(id_1 \| r_1) + \alpha_2 H(id_2 \| r_2)}). \end{aligned}$$

–non-malleability. Given m_1 and m_2 with identifiers id_1 and id_2, and their signatures $\sigma_1 = (s_1, r_1)$ and $\sigma_2 = (s_2, r_2)$ generated by user t, an adversary who does not possess the private key sk_t, should not be able to generate a valid signature on a block $m' = \alpha_1 m_1 + \alpha_2 m_2$ by combining σ_1 and σ_2, where α_1 and α_2 are random values in $\mathbb{Z}_q$. To prove this, we assume that the adversary generates $\sigma' = (s', r')$ on block m' with identifier id' by combining $\sigma_1 = (s_1, r_1)$ and $\sigma_2 = (s_2, r_2)$ as $s' = s_1^{\alpha_1} s_2^{\alpha_2}$ and $r' = r_1^{\alpha_1} r_2^{\alpha_2}$. So, we can write the following equalities:

$$r' = r_1^{\alpha_1} r_2^{\alpha_2} = h^{e_1 \alpha_1} h^{e_2 \alpha_2} = h^{e_1 \alpha_1 + e_2 \alpha_2} \Rightarrow e' = e_1 \alpha_1 + e_2 \alpha_2.$$

$$\begin{aligned} s' &= s_1^{\alpha_1} s_2^{\alpha_2} \\ &= h^{\alpha_1 sk_t (m_1 + e_1 + H(id_1 \| r_1))} h^{\alpha_2 sk_t (m_2 + e_2 + H(id_2 \| r_2))} \\ &= h^{sk_t \left(\alpha_1 m_1 + \alpha_2 m_2 + \alpha_1 e_1 + \alpha_2 e_2 + \alpha_1 H(id_1 \| r_1) + \alpha_2 H(id_2 \| r_2)\right)} \\ &= h^{sk_t \left(m' + e' + \alpha_1 H(id_1 \| r_1) + \alpha_2 H(id_2 \| r_2)\right)}. \end{aligned}$$

$$\Rightarrow \quad H(id' \parallel r') = \alpha_1 H(id_1 \parallel r_1) + \alpha_2 H(id_2 \parallel r_2) \quad (6)$$

Thus, the adversary should find id' in a way that it satisfies Equation 6 which contradicts the one-way property of hash function H.

Here, we have proved the blockless verifiability and non-malleability properties for a second-level signature. Proving these properties for a first-level signature is trivially the same as the arguments above and therefore HAS_{uni} is a homomorphic authenticable proxy re-signature scheme. □

5 CoRPA

5.1 Construction

In this section we employ HAS_{uni}, the new homomorphic authenticable proxy re-signature scheme proposed in previous section, to build a Collusion Resistant Public Auditing scheme (CoRPA). CoRPA includes eight algorithms which are described in Fig. 3: **Setup**, **KeyGen**, **ReKey**, **Sign**, **ReSign**, **Challenge**, **ProofGen**, **Proof Verify**.

In **KeyGen**, each user in the group U, generates his public/private key pair. Also, the system public/private key is generated by the key generation center where the system private key is only held by the KGC.

In **ReKey**, the KGC, based on the system private key and users' public keys, generates d re-signature keys for d members in U and publishes them publicly. The re-signature key for user $i \in U$, $rk_{i \to S}$, converts a signature from user i to a system signature. Since for generating $rk_{i \to S}$, the KGC only needs the public key of user i, no private channel between entities is assumed as oppose to [10]. Also, collusion of the server and any of users reveals no information about the system private key or the private key of other users as proved in Theorem 1.

In **Sign**, user i using his private key, signs the block he has created/modified and sends the vector $\{m_j, id_j, \sigma_{ij}\}$ to the server, where σ_{ij} is the first-level signature from user i on (m_j, id_j). Also, in order to prevent the server from applying replace/replay attacks as in [20] or the attack in [13], the group manager signs and sends the pair (id_j, r_{ij}) to the TPA for each vector $\{m_j, id_j, \sigma_{ij}\}$ outsourced to the server, where r_{ij} is a part of σ_{ij}. We should mention that **Sign** algorithm in CoRPA can be performed in two **online/offline** phases as the schemes in [5], [6]. Since the term $r_{ij} = h^{e_{ij}}$ in signature generation does not depend on the message, it can be calculated offline, for example when the resource constrained user is connected to power. Hence, the online phase of **Sign** algorithm is just calculating the linear combination $s_{ij} = k_i(m_j + e_{ij} + H_1(id_j \parallel r_{ij}))(mod q)$ which has low overhead and can be executed very fast.

In **ReSign**, the server translates the received signature to a second-level one and saves (m_j, id_j, σ_j) in his storage. Since all the translated-signatures are system signatures, the TPA cannot distinguish the identity of users who signed shared data blocks and the scheme preserves identity privacy. This signature translation done by server also enables efficient user revocation which will be discussed in Subsection 5.1.1.

In order to audit the data integrity, a challenge-response protocol is executed between the server and the TPA. The TPA first chooses random block indices and sends $(\{(j, y_j)\}_{j \in J}, c_1, c_2)$ to the server in **Challenge** (see Fig. 3). Then the server runs **ProofGen** to compute the proof as $P = e(c_1, s).c_2{}^{-\mu}$ where μ and s, as described in Fig. 3, are aggregated challenged blocks and their signatures, respectively. Finally the proof correctness is checked by the TPA via **Proof Verify**. We note that the proof P sent to the TPA is a combination of μ and s. As the only information the TPA gains within this protocol is P, the data privacy is preserved in addition to the signer identity privacy which will be formally proved in Subsection 5.2. Furthermore, if necessary, the TPA can prove to the server that c_1 and c_2 have equal discrete logarithms by Chaum-Pederson protocol [21] and send this proof of equality along with the challenge to the server, as done in [7].

5.1.1 *User Revocation*

In CoRPA, when a user leaves the group or misbehaves, he can be revoked from the group efficiently and securely which are discussed in the following paragraphs:

Efficiency. When a user is revoked, the signatures generated by this user become invalid to the group and should be re-signed. But since in CoRPA all signatures are previously translated to the system signature, this re-signing operation is not needed. So, to revoke a user, the data owner (group manager) just sends the revoked user's id to the server and the server updates his revocation list (RL). Therefore, user revocation in our mechanism is ultra-efficient.

Setup. Consider the map $e : G_1 \times G_1 \to G_2$ with G_1 and G_2 as cyclic groups of prime order q, and g,h as generators of G_1. The public parameters are $(e, q, G_1, G_2, g, h, H_1)$, where $H_1 : \{0,1\}^* \to \mathbb{Z}_q$. Shared data is described as $M = (m_1, ..., m_n)$ where $m_i \in \mathbb{Z}_q$ are the blocks that M is divided to. Also, the group of d users who share the data M in denoted by U.
KeyGen. User $i \in U$, on input the global parameters $(e, q, G_1, G_2, g, h, H_1)$, selects a random $k_i \in \mathbb{Z}_q$ and generates the public key as $pk_i = (g^{k_i}, h^{1/k_i})$ and private key as $sk_i = (k_i, h^{k_i})$. The group manager also creates and signs the group users list, UL, which is public to all entities. The system public/private key pair $(pk_S = g^k, sk_S = k)$ is also generated by the Key Generation Center (KGC) where k is randomly chosen from $\mathbb{Z}_q$.
ReKey. In our protocol, the re-signature key for user $i \in U$, $rk_{i\to S}$, converts a signature from user i to a system signature. The trusted KGC, on input pk_i and sk_S, generates the re-signature key for user i as $rk_{i\to S} = (pk_i)^{sk_S} = (h^{1/k_i})^k = h^{k/k_i}$. The d re-signature keys for d users generated by the KGC in this step are published publicly.
Sign. To sign a block m_j with block identifier id_j, user i with $sk_i = k_i$, selects $e_{ij} \in_r \mathbb{Z}_q$ and outputs first-level signature $\sigma_{ij} = (s_{ij}, r_{ij})$ as follows: $r_{ij} = h^{e_{ij}}$ and $s_{ij} = k_i(m_j + e_{ij} + H_1(id_j \parallel r_{ij}))(mod q)$. User i, after signing the new/modified block m_j, sends (m_j, id_j, σ_{ij}) to be re-signed and stored at the server. Furthermore, the group manager signs and sends the pair (id_j, r_{ij}) to the TPA for each vector $\{m_j, id_j, \sigma_{ij}\}$ outsourced to the server.
ReSign. The server, given re-signature key $rk_{i\to S}$, public key pk_i, first-level signature σ_{ij}, and block m_j with block identifier id_j, first checks that Verify(pk_i,m_j,id_j,σ_{ij})= 1. If the verification does not hold, output $\perp$; otherwise computes the second-level signature $\sigma_j = (s_j, r_j)$, where $r_j = r_{ij}$ and $s_j = (rk_{i\to S})^{s_{ij}} = (h^{k/k_i})^{k_i(m_j+e_j+H_1(id_j\|r_j))} = h^{k(m_j+e_j+H_1(id_j\|r_j))}(mod q)$ and stores (m_j, id_j, σ_j) in his storage.
Challenge. To challenge shared data integrity, the TPA does the following:

1) Chooses a random c-element subset $J \subset [1, n]$ as the block indices to be challenged in the auditing process and for each $j \in J$ chooses a random value $y_j \in \mathbb{Z}_{q'}$.
2) Picks random $\rho \in \mathbb{Z}_q$ and computes $c_1 = g^\rho$ and $c_2 = Z^\rho$, where $Z = e(pk_S, h)$.
3) Sends the auditing message $(\{(j, y_j)\}_{j\in J}, c_1, c_2)$ to the server.

ProofGen. The server, based on the received challenge $(\{(j, y_j)\}_{j\in J}, c_1, c_2)$, generates an auditing proof through the following procedure:

1) Computes $\mu = \sum_{j\in J} y_j m_j$, as the combination of the challenged blocks.
2) Aggregates the signatures as $s = \prod_{j\in J} s_j^{y_j}$.
3) Computes $P = e(c_1, s).{c_2}^{-\mu}$ and sends back P as the auditing proof to the TPA.

ProofVerify. The TPA verifies the server's proof P through Equation 7. The verification passes if the equation holds and fails otherwise. The values $\{(id_j, r_j)\}_{j\in J}$ are known by the TPA since the group manager sends the latest version of these values to the TPA.

$$P \overset{?}{=} e\Bigg(pk_S, \prod_{j\in J}\left(r_j h^{H_1(id_j\|r_j)}\right)^{y_j}\Bigg)^\rho \tag{7}$$

Fig. 3. Details of CoRPA

Security. User revocation in our mechanism is secure due to the following statements:

- The revoked user can no longer access and modify shared data, because the server by checking the RL prevents the revoked user from accessing or uploading the data.
- If the server and the revoked user collude, they cannot learn any information about the system private key or the private key of other users as proved in Theorem 1. However, because the server still possess the re-signature key related to the revoked user, the server and the revoked user together are able to arbitrarily modify shared data blocks and generate valid signatures on them; but we claim that the modified blocks will not be verified by the TPA. Because the values $\{(id_j, r_j)\}_{j\in J}$ used by the TPA in verification, are sent by the group manager each time a data block is modified. When the server and the revoked user collude and modify a data block, since the revoked user is not in the authorized users list, the group manager does not send updated (id_j, r_j) to the TPA and the collusion fails.

5.1.2 *Dynamic Data Operation*

All users in group U can modify the shared data blocks including insert a new block, and delete or update an existing block. To support dynamic data operations efficiently, there are some considerations about the block identifiers which are discussed here. As the block signatures depend on the block identifiers, using the index of a block as its identifier is not efficient. More precisely, because by inserting or deleting a special block, the indices of all the blocks after this block change, the signature on all these blocks needs to be re-computed. In order to solve this issue, the concept of virtual index was introduced [8], [17]. Specifically, the identifier of block m_j is defined as $id_j = \{v_j, h_j\}$, where v_j and h_j are defined as follows:

- v_j is the virtual index of the block which determines the blocks orders. The first time the data is created by the data owner, the value $v_j = j.\delta$ is assigned to block m_j, where δ is a system parameter determined by the data owner. But what makes the virtual index different with the normal one, is that when a user performs insert or delete operations, the index of the blocks after the modified block does not change which will be explained in the **Modify** algorithm below.
- $h_j = H_2(m_j \parallel v_j)$, where $H_2 : \{0,1\}^* \to \mathbb{Z}_{q'}$ is a collision-resistant hash function and q' is much smaller prime than q.

In the following, we describe the details of **Modify** algorithm in CoRPA.

Modify. Here Ins, Del and Up denote insert, delete and update operations, respectively.

- **Insert.** To insert a new block m'_j into the shared data, user i first computes $id'_j = \{v'_j, h'_j\}$, where $v'_j = (v_{j-1} + v_j)/2$, and $h'_j = H_2(m'_j \parallel v'_j)$. Next, he generates the signature σ'_{ij} of m'_j and sends $\{m'_j, id'_j, \sigma'_{ij}\}$ to the server. Also, the group manager signs and sends $\{Ins, j, (id'_j, r'_{ij})\}$ to the TPA (r'_{ij} is a part of σ'_{ij}). The TPA inserts (id'_j, r'_{ij}) in the jth position of his list L. Obviously, the identifiers of other blocks are not changed and so there is no need to update signatures on other blocks.
- **Delete.** The user sends a request to delete a block m_j, id_j and its signature σ_j from the data stored at the server. Also, the group manager signs and sends $\{Del, j\}$ to the TPA. The TPA deletes (id_j, r_{ij}) from the jth position of L. Again, the identifiers of other blocks and their signatures remain unchanged.
- **Update.** To update the j-th block with a new value m'_j, user i first computes $id'_j = \{v_j, h'_j\}$, where the virtual index v_j remains the same and h'_j is computed as $h'_j = H_2(m'_j \parallel v_j)$. Next, he generates the signature σ'_{ij} of m'_j and sends $\{m'_j, id'_j, \sigma'_{ij}\}$ to the server. Also, the group manager signs and sends $\{Up, j, (id'_j, r'_{ij})\}$ to the TPA. The TPA updates the jth position of L with (id'_j, r'_{ij}). Again, the identifiers of other blocks and their signatures remain unchanged.

5.1.3 *Batch Auditing*

We argue that in CoRPA, the TPA can handle multiple auditing tasks simultaneously. Suppose that the TPA has received T auditing requests from T different groups of users in a short time period. Obviously, performing these requests one by one is inefficient and the TPA should be able to perform all auditing tasks via a single equation which is named batch auditing.

More precisely, consider T auditing tasks received by the TPA from T groups of users $U_1, ..., U_T$ who share the data $M_1, ..., M_T$, respectively. The TPA, for each auditing task $t \in [1, T]$, picks random $\rho_t \in \mathbb{Z}_q$ and generates a challenge $(\{(j, y_{j|t})\}_{j \in J_t}, c_{1|t}, c_{2|t})$. The server, upon receiving the challenge for $t \in [1, T]$, generates an auditing proof $P_t = e(c_{1|t}, s_t).{c_{2|t}}^{-\mu_t}$ and sends it back to the TPA. The TPA, aggregates T received proofs and verifies them via one equation as below:

$$\prod_{t \in T} P_t \overset{?}{=} e(pk_S, \prod_{t \in T} \Big(\prod_{j \in J_t} \big(r_{j|t} h^{H_1(id_{j|t} \| r_{j|t})}\big)^{y_{j|t}} \Big)^{\rho_t}) \tag{8}$$

The above equation holds iff all the auditing proofs $P_1, ..., P_T$ are correct, which can be shown using bilinear maps properties as below:

$$\begin{aligned} \prod_{t \in T} P_t &= \prod_{t \in T} e(pk_S, \Big(\prod_{j \in J_t} \big(r_{j|t} h^{H_1(id_{j|t} \| r_{j|t})}\big)^{y_{j|t}} \Big)^{\rho_t}) \\ &= e(pk_S, \prod_{t \in T} \Big(\prod_{j \in J_t} \big(r_{j|t} h^{H_1(id_{j|t} \| r_{j|t})}\big)^{y_{j|t}} \Big)^{\rho_t}). \end{aligned} \tag{9}$$

According to Equation 8, for verifying T auditing tasks, instead of T pairing operations, the TPA only computes one pairing via batch auditing. Therefore, batch auditing brings a great efficiency enhancement for multiple data verification requests.

5.2 Security Analysis of CoRPA

In this section we first prove the protocol correctness. Next, we investigate two security properties of CoRPA: security against server (soundness) and privacy against TPA which are proved via Theorems 3 and 4, respectively.

Correctness. If both the cloud server and the public verifer are honest, the cloud server can pass the auditing in **ProofVerify** which is shown as below:

$$\begin{aligned} P = e(c_1, s).{c_2}^{-\mu} &= \frac{e(c_1, s)}{e(pk_S, h)^{\rho \sum_{j \in J} y_j m_j}} \\ &= \frac{e(g^\rho, \prod_{j \in J} s_j^{y_j})}{e(g^k, h)^{\rho \sum_{j \in J} y_j m_j}} \\ &= \frac{e(g^\rho, \prod_{j \in J} s_j^{y_j})}{e(g^\rho, h^k)^{\sum_{j \in J} y_j m_j}} \\ &= \frac{e(g^\rho, \prod_{j \in J} s_j^{y_j})}{e(g^\rho, \prod_{j \in J} (h^{k m_j})^{y_j})} \\ &= e(g^\rho, \prod_{j \in J} \frac{s_j^{y_j}}{(h^{k m_j})^{y_j}}) \\ &= e(g^\rho, \prod_{j \in J} \frac{h^{k(m_j + e_j + H_1(id_j \| r_j))^{y_j}}}{(h^{k m_j})^{y_j}}) \\ &= e\Big(g^\rho, \prod_{j \in J} (h^{k(e_j + H_1(id_j \| r_j))})^{y_j}\Big) \\ &= e\Big(g^k, \prod_{j \in J} (r_j h^{H_1(id_j \| r_j)})^{y_j}\Big)^\rho \\ &= e\Big(pk_S, \prod_{j \in J} \big(r_j h^{H_1(id_j \| r_j)}\big)^{y_j}\Big)^\rho \end{aligned} \tag{10}$$

Theorem 3. Security against Server (Soundness): *CoRPA is secure against an untrusted server under the DL assumption. In other words, the server can pass the verification iff the stored shared data has been preserved intact.*

Proof. A valid response to challenge $(\{(j, y_j)\}_{j \in J}, c_1, c_2)$ from the TPA is in the form of $P = e(c_1, s).{c_2}^{-\mu}$, where $s = \prod_{j \in J} s_j^{y_j}$, $\mu = \sum_{j \in J} y_j m_j$ and $\{m_j\}_{j \in J}$ are the correct data blocks outsourced by users. But here we assume that the server cheats and sends the proof as $P' = e(c_1, s).{c_2}^{-\mu'}$, where $\mu' = \sum_{j \in J} y_j m'_j$ and at least one data block m'_j is manipulated.

Now we show that if the server can fool the TPA and invalid proof passes the verification, it will be easy to solve the discrete logarithm problem. As both P and P' pass the verification, we have:

$$P = e\Big(pk_S, \prod_{j \in J} \big(r_j h^{H_1(id_j \| r_j)}\big)^{y_j}\Big)^\rho \tag{11}$$

$$P' = e\Big(pk_S, \prod_{j \in J} \big(r_j h^{H_1(id_j \| r_j)}\big)^{y_j}\Big)^\rho \tag{12}$$

Since the values $\{(id_j, r_j)\}_{j \in J}$ are replaced by the TPA in the verification equation, the RHS of both equations are equal, so we have $P = P'$ which can be written as:

$$\begin{aligned} e(c_1, s).{c_2}^{-\mu} &= e(c_1, s).{c_2}^{-\mu'}, c_2 = e(pk_S, h)^\rho \\ \Rightarrow e(c_1, s).e(pk_S, h^{-\mu\rho}) &= e(c_1, s).e(pk_S, h^{-\mu'\rho}) \end{aligned} \tag{13}$$

From the above equality we can learn that $h^{-\mu\rho} = h^{-\mu'\rho} \Rightarrow (h^{\rho})^{\triangle\mu} = 1$, where $\triangle\ \mu = \mu - \mu' \neq 0$. Also, as was defined in Fig. 3, g,h are generators of cyclic group G_1. So, there exists $x \in \mathbb{Z}_q$ such that $h = g^x$. Having $g, h \in G_1$, h^{ρ} can be written as $h^{\rho} = g^{\varsigma} h^{\tau}$, where ς and τ are random values in $\mathbb{Z}_q$. Therefore, we can compute x as follows:

$$1 = (h^{\rho})^{\triangle\mu} = (g^{\varsigma} h^{\tau})^{\triangle\mu} = g^{\varsigma\triangle\mu} h^{\tau\triangle\mu} \Rightarrow h = g^{-\frac{\varsigma\triangle\mu}{\tau\triangle\mu}}$$
$$\Rightarrow x = -\frac{\varsigma\triangle\ \mu}{\tau\triangle\ \mu}. \quad (14)$$

where the dominator $\tau\triangle\ \mu$ is not zero with probability $(1-1/q)$ ($\tau \in \mathbb{Z}_q$ is zero with probability $1/q$). Therefore, if the invalid proof generated by the cheating server passes the verification, we can solve the discrete logarithm problem with probability $1-1/q$, which contradicts the hardness assumption of DL in G_1 and completes the proof. □

Theorem 4. Privacy against TPA: *During the protocol execution, no information of the data content or the signer identities is leaked to the TPA.*

Proof. For data privacy proof, we show that a simulator $\mathcal{S}$ with blackbox access to the TPA can simulate the **ProofGen** algorithm without the need to know data blocks or their signatures as follows:

In response to challenge $(\{(j, y_j)\}_{j\in J}, c_1, c_2)$ received from the TPA, $\mathcal{S}$ extracts $\rho, \{(id_j, r_j)\}_{j\in J}$ from the public verifier and outputs $P{=}e\left(pk_S, \prod_{j\in J}\left(r_j h^{H_1(id_j\|r_j)}\right)^{y_j}\right)^{\rho}$, where pk_S is the system public key. It can be seen that the proof generated by simulator $\mathcal{S}$ is valid and also it contains no information of the data blocks. So, CoRPA preserves data privacy.

For identity privacy, as all signatures stored at the cloud server are in the form of system signature, the TPA only needs public key of the system pk_S in the auditing process and no information of the signer identities is leaked to the public verifier. □

6 Performance

In this section, we first analyze the computation, communication and storage overheads of CoRPA and make a comparison with the state of the art. Next, we provide the experimental evaluations of the protocol.

6.1 Overhead Analysis

Here we denote modular multiplications, exponentiations and pairings by Mul, Exp, and $Pair$, respectively. Also, the index of the operation denotes the group that the operation is defined in, for example Mul_{G_1} means multiplication over G_1. Furthermore, as additions have negligible cost in comparison to other operations, they are omitted from overhead analysis.

Computation Overhead. We present the computation overhead for the KGC, the users, the cloud server and the TPA. The dominated computation of the KGC, is generating the re-signature keys for d users of the group U. So, the main computation cost of the KGC is $d.Exp_{G_1}$.

To compute a signature on a block of shared data, a user in the group U has the cost $1H + 1Mul_{\mathbb{Z}_q} + 1Exp_{G_1}$. Here we mention that the term $r_{ij} = h^{e_{ij}}$ in signature generation does not depend on the message and can be calculated offline. Hence, the online phase of **Sign** algorithm is just calculating the linear combination $s_{ij} = k_i(m_j + e_{ij} + H_1(id_j \parallel r_{ij}))(mod q)$ which has the cost $1H + 1Mul_{\mathbb{Z}_q}$.

The main computation cost of the TPA is generating a challenge to audit the data integrity and verifying the proof returned by the server. The pairing $Z = e(pk_S, h)$ is only computed once and is not re-computed in each challenge, thus the TPA's cost for each challenge is $1Exp_{G_1} + 1Exp_{G_2}$, for computing c_1 and c_2, respectively. Also, due to Equation 7, the term $r_j h^{H_1(id_j\|r_j)}$ is not dependent to the challenge and can be computed offline for the entire file in advance. Therefore, the online cost of proof verification is $1Exp_{G_2} + 1Pair + cExp_{G_1} + (c-1)Mul_{G_1}$.

The functions done by the cloud server include translation of the signatures and generating a proof in response to the TPA. For signature translation, the server first verifies the original signature and then re-signs it using the related re-signature key. So, the overhead of signature translation for the server is $2Exp_{G_1} + 1H + 1Mul_{G_1} + 2Pair$. This overhead can be further reduced to only one Exp_{G_1} by omitting the verification and only re-signing the original signature, because the auditing done by the TPA ensures the signature's correctness. Also, the proof generation cost for the server is equal to $(c-1)Mul_{\mathbb{Z}_q} + cExp_{G_1} + (c-1)Mul_{G_1} + 1Exp_{G_2} + 1Mul_{G_2} + 1Pair$.

Communication Overhead. Since in CoRPA there is no need to re-sign the blocks signed by the revoked user, user revocation has no communication overhead on users. The communication cost of the TPA and the server is introduced by challenge and proof in the auditing protocol, respectively. For each challenge $(\{(j, y_j)\}_{j\in J}, c_1, c_2)$, where y_j, c_1, and c_2 are elements of $\mathbb{Z}_{q'}, G_1$, and G_2, the communication cost is $c(|n| + |q'|) + 2|q|$ bits, in which $|q|$ and $|q'|$ are the element lengths of $\mathbb{Z}_q$ and $\mathbb{Z}_{q'}$, and $|n|$ is the length of an index. Also, the communication cost for each auditing proof $P = e(c_1, s).c_2{}^{-\mu}$ is equal to $|q|$ bits.

Storage Overhead. In CoRPA, the data users store nothing since the data blocks and their signatures are stored at the cloud server. For each block of shared data, the server stores (m_j, id_j, σ_j) in his storage, where the lengths of m_j,id_j and σ_j are $|q|$, $|q'|$ and $2|q|$, respectively (in $\sigma_j = (s_j, r_j)$, s_j and r_j are elements of G_1). So, the storage cost of the cloud server is $n(3|q| + |q'|)$ where n is the total number of shared data blocks. Also, in CoRPA the TPA saves a term $r_j h^{H_1(id_j\|r_j)}$ for each data block outsourced to the server which has a total storage cost of $n.|q|$ for n data blocks. We note that this list stored by the TPA, was used in the verification to make the protocol secure against attacks proposed in [13], [20].

Table 2 presents a summary of the total overheads in CoRPA discussed above and makes a comparison with the state of the art.

6.2 Experimental Results

In this part, we evaluate the efficiency of our mechanism and make a comparison with Oruta [8] and Panda [10]. We implemented the three schemes on a personal computer (Intel I5-3470 3.20 GHz processor, 4 GB memory and the Windows 7 operating system) using the MIRACL library [22]. We assume $|q| = 160$ bits, $|q'| = 80$ bits, $n = 1,000,000$ and $|n| = 20$ bits, where n is the number of blocks that shared data M is divided to. Also, the size of M is assumed to be 2 GB. Furthermore, due to

TABLE 2
Performance comparison

Scheme / Metric	Oruta [8]	Panda [10]	CoRPA
Sign. Gen. Complexity	$1H+(d+k)Mul_{G_1}+(2d+k)Exp_{G_1}$	$1H+1Mul_{G_1}+2Exp_{G_1}$	Online: $1H+1Mul_{\mathbb{Z}_q}$ Offline: $1Exp_{G_1}$
Server Comp. Complexity	$kH+(ck)Mul_{\mathbb{Z}_q}+(k+dc)Exp_{G_1}+(dc)Mul_{G_1}$	$cMul_{\mathbb{Z}_q}+cExp_{G_1}+cMul_{G_1}$	$cMul_{\mathbb{Z}_q}+cExp_{G_1}+cMul_{G_1}+1Exp_{G_2}+1Mul_{G_2}+1Pair$
Verifier Comp. Complexity	$cH+dMul_{G_2}+(d+2)Pair+(2k+c)Exp_{G_1}+(2k+c)Mul_{G_1}$	$cH+dMul_{G_2}+(d+1)Pair+(c+d)Exp_{G_1}+(c+2d)Mul_{G_1}$	$2Exp_{G_2}+1Pair+cExp_{G_1}+cMul_{G_1}$
Comm. Complexity	$(2k+d)\|q\|+c(2\|q'\|+\|n\|)$	$2d\|q\|+c(2\|q'\|+\|n\|)$	$3\|q\|+c(\|q'\|+\|n\|)$

Parameters d, k and c denote size of the group U, number of elements per block and number of challenged blocks in an auditing task, respectively.

[2], to have detection probability greater than 99%, we choose $c = 460$. As can be seen in Figs. 4, 5 and 6, by increasing the value of d, the distance between CoRPA and the other two schemes increases and the efficiency of our mechanism becomes more clear. Also, because in CoRPA, as oppose to the other two schemes, signature generation and the server and verifier computation and communication costs are not dependent on the number of group users d, our mechanism supports large dynamic groups efficiently.

6.2.1 *Performance of Signature Generation*

Fig. 4 illustrates the time needed for generating signature on a block in CoRPA versus the number of users in group U which is assumed to change in the interval $d \in [1, 10]$ for better visibility. Specifically, when $d = 10$, a user in CoRPA needs 3.815 milliseconds to generate a signature on a block of shared data, while the relevant times for Oruta and Panda are 78.835 and 7.555, respectively. We also note that in Oruta [8], the authors have used block sectoring technique to reduce the storage space of ring signature on a block. Here we put $k = 1$ (no sectoring), to fairly compare the signature generation time of the schemes in Fig. 4. Also for CoRPA, we have considered the total cost of signature generation (Online+Offline) in the plot.

6.2.2 *Performance of Auditing*

As shown in Fig. 5 and 6, the communication cost and auditing time of Oruta and Panda linearly increases with the number of users in the group. While in CoRPA these parameters remain fixed as the number of group users increases. Specifically, when $d = 20$, our mechanism can end an auditing task with only 5.79 KB and 1743 milliseconds when $c = 460$. But under our testbed, the relevant values for Oruta are 10.79 KB and 2015 milliseconds and for Panda are 11.15 KB and 2072 milliseconds when $d = 20$ and $c = 460$.

Batch Auditing. As discussed in Section 5, when the TPA receives multiple auditing tasks in a short time period, it can perform all tasks simultaneously via batch auditing which significantly decreases the number of pairing operations from T pairing to only one. Fig. 7 demonstrates the performance of batch auditing in comparison to independent auditing when $c = 460$ and for any arbitrary value of d. As can be seen in this figure, with batch auditing, the average time required for each auditing task is efficiently reduced. For example, when the number of simultaneous tasks is $T = 20$, the average time per auditing task in batch auditing is about 1732 milliseconds, while with independent auditing this average is about 1743 milliseconds. Therefore, for $T = 20$, the TPA can save totally around $20 \times 11 = 220$ milliseconds via batch auditing .

7 Related Work

Remote data integrity checking which includes the concepts of Provable Data Possession (PDP) and Proof of Retrievability (POR), has been one of the hottest research interests in the field of cloud computing in recent years. PDP, allows a third party to verify integrity of the data stored at an untrusted server on behalf of data owners without the need to retrieve data from the cloud server. POR techniques, in addition to data integrity auditing, prevents data corruption by using forward error-correcting codes, remotely. Ateniese et al. [2] and Jules et al. [23], proposed the first PDP and POR schemes using RSA-based homomorphic tags, respectively. However, the use of RSA numbering in RSA-based homomorphic tags, results in expensive communication and computation costs and make the schemes inefficient. Moving a step forward, Shacham and waters [3], proposed BLS-based homomorphic tags [24] and designed an efficient POR scheme based on it.

To support dynamic data operations efficiently, Ateniese et al. [25] proposed a symmetric-key PDP scheme which is not publicly verifiable and also only supports limitted number of verification requests. Wang et al. [26] utilized Merkle Hash Tree to introduce a public auditing mechanism with fully dynamic operations. Erway et al. [27] also proposed another dynamic PDP scheme based on the rank information. Later, Zhu et al. [17], exploited index hash tables to support dynamic data operations. They also employed the block sectoring technique to reduce the storage overhead of signatures. Also, [28] and [29] are other very recent dynamic PDP proposals. However, most POR schemes fail to support efficient dynamic data operations. Recently, Cash et al. [30] proposed the first dynamic POR scheme by utilizing oblivious RAM which is private and do not support public verification.

To audit the integrity of the data which is shared among a group of users, Wang el al. [8] proposed a scheme which exploits ring signatures to protect the user's identity privacy against the public verifier. However, their scheme does not support dynamic groups and efficient user revocation. Also, the large signature size and auditing cost in this mechanism, makes it inefficient for practical scenarios. Later, Wang et al. [10] enhanced their previous public auditing scheme using proxy re-signatures to support efficient user revocation. However, it is assumed that there is no collusion between the cloud server and any user. If the server and a revoked user collude, they can easily recover the private key of other users. Also, their scheme does not provide data and identity privacy which is an essential feature in shared data auditing mechanisms. Furthermore, the auditing cost of the scheme is linear to the number of users which makes it inefficient

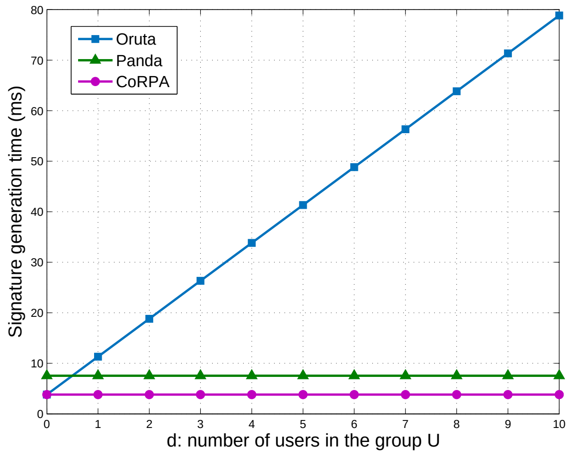

Fig. 4. Impact of d on Signature generation time, where $k = 1$.

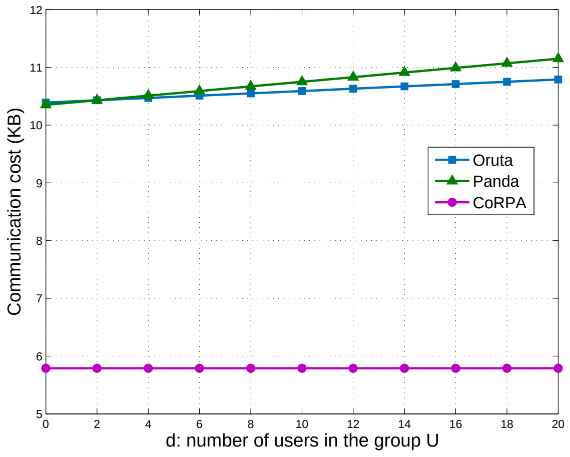

Fig. 6. Impact of d on communication cost, where $k = 1$ and $c = 460$.

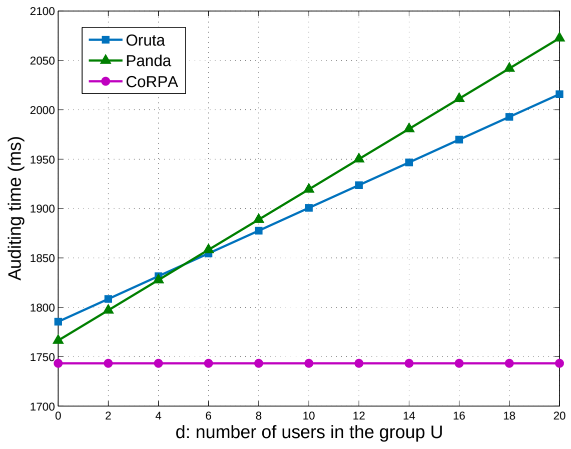

Fig. 5. Impact of d on auditing time, where $k = 1$ and $c = 460$.

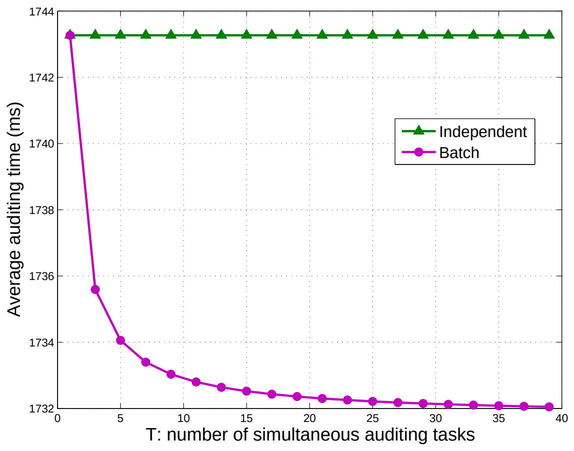

Fig. 7. Impact of T on the efficiency of batch auditing, where $c = 460$.

for large groups. Recently, Yuan and Yu proposed a shared data auditing scheme with collusion-resistant user revocation utilizing polynomial-based authentication tags [12]. However, since the TPA is involved in the user revocation, communication and computation costs of revocation are increased in comparison to [10]. Also, the scheme does not provide identity privacy. Jiang et al. [14] also considered the problem of secure user revocation where by employing the group signatures [15], they prevent the collusion of cloud and revoked users. However, the expensive computation cost of generating group signatures and also costly auditing operations, make the scheme inefficient. Furthermore, no formal proof is provided in the paper to show that the scheme is collusion resistant [14].

8 Conclusion and Future Work

In this paper, we proposed a new public shared data auditing scheme that provides identity privacy and collusion resistant user revocation, simultaneously. By employing a collusion resistant proxy re-signature scheme, the cloud server translates all the received signatures and saves (block,translated-signature) pair in his storage. Since all the signatures stored at the server are translated to system signatures, no one can distinguish who the signer of a special block is, and the scheme preserves identity privacy. It also enables efficient user revocation, because re-signing of the blocks signed by the revoked user is not needed. Meanwhile, the server who possess the re-signature keys, is not able to independently generate a valid signature on an arbitrary block on behalf of users or system even if he collude with one of the group users (e.g. a revoked user), which was formally proved in the paper. Moreover, our protocol supports large dynamic group of users, batch verification of multiple auditing tasks and fully dynamic data operations, efficiently. Overhead analysis and experimental results showed that computation and communication overheads of our protocol are constant and do not change with the number of group users; accordingly, our public auditing scheme can achieve fine efficiency.

References

[1] Cloud Security Alliance. (2010). Top threats to cloud computing. [Online]. Available: http://www.cloudsecurityalliance.org

[2] G. Ateniese, R. Burns, R. Curtmola, J. Herring, L. Kissner, Z. Peterson, and D. Song, "Provable data possession at untrusted stores," in *Proceedings of the 14th ACM conference on Computer and communications security, ACM 2007.* Alexandria, Virginia, USA: ACM, October 29 - November 02 2007, pp. 598–609.

[3] H. Shacham and B. Waters, "Compact proofs of retrievability," in *International Conference on the Theory and Application of Cryptology and Information Security.* Springer, 2008, pp. 90–107.

[4] Q. Wang, C. Wang, K. Ren, W. Lou, and J. Li, "Enabling public auditability and data dynamics for storage security in cloud computing," *IEEE transactions on parallel and distributed systems*, vol. 22, no. 5, pp. 847–859, 2011.

[5] J. Li, L. Zhang, J. K. Liu, H. Qian, and Z. Dong, "Privacy-preserving public auditing protocol for low-performance end devices in cloud," *IEEE Transactions on Information Forensics and Security*, vol. 11, no. 11, pp. 2572–2583, 2016.

[6] Y. Wang, Q. Wu, B. Qin, S. Tang, and W. Susilo, "Online/offline provable data possession," *IEEE Transactions on Information Forensics and Security*, vol. 12, no. 5, pp. 1182–1194, 2017.

[7] Y. Yu, M. H. Au, G. Ateniese, X. Huang, W. Susilo, Y. Dai, and G. Min, "Identity-based remote data integrity checking with perfect data privacy preserving for cloud storage," *IEEE Transactions on Information Forensics and Security*, vol. 12, no. 4, pp. 767–778, 2017.

[8] B. Wang, B. Li, and H. Li, "Oruta: privacy-preserving public auditing for shared data in the cloud," *IEEE transactions on cloud computing*, vol. 2, no. 1, pp. 43–56, 2014.

[9] D. Boneh, C. Gentry, B. Lynn, and H. Shacham, "Aggregate and verifiably encrypted signatures from bilinear maps," in *International Conference on the Theory and Applications of Cryptographic Techniques: Advances in Cryptology– EUROCRYPT'03*. Warsaw, Poland: Springer, May 4-8 2003, pp. 416–432.

[10] B. Wang, B. Li, and H. Li, "Panda: public auditing for shared data with efficient user revocation in the cloud," *IEEE Transactions on services computing*, vol. 8, no. 1, pp. 92–106, 2015.

[11] M. Blaze, G. Bleumer, and M. Strauss, "Divertible protocols and atomic proxy cryptography," in *Advances in Cryptology– EUROCRYPT'98*. Espoo, Finland: Springer, May 31 - June 4 1998, pp. 127–144.

[12] J. Yuan and S. Yu, "Public integrity auditing for dynamic data sharing with multiuser modification," *IEEE Transactions on Information Forensics and Security*, vol. 10, no. 8, pp. 1717–1726, 2015.

[13] Y. Yu, Y. Li, J. Ni, G. Yang, Y. Mu, and W. Susilo, "Comments on public integrity auditing for dynamic data sharing with multiuser modification," *IEEE Transactions on Information Forensics and Security*, vol. 11, no. 3, pp. 658–659, 2016.

[14] T. Jiang, X. Chen, and J. Ma, "Public integrity auditing for shared dynamic cloud data with group user revocation," *IEEE Transactions on Computers*, vol. 65, no. 8, pp. 2363–2373, 2016.

[15] D. Boneh and H. Shacham, "Group signatures with verifier-local revocation," in *Proceedings of the 11th ACM conference on Computer and communications security*. ACM, 2004, pp. 168–177.

[16] G. Ateniese and S. Hohenberger, "Proxy re-signatures: new definitions, algorithms, and applications," in *Proceedings of the 12th ACM conference on Computer and communications security*. ACM, 2005, pp. 310–319.

[17] Y. Zhu, H. Wang, Z. Hu, G.-J. Ahn, H. Hu, and S. S. Yau, "Dynamic audit services for integrity verification of outsourced storages in clouds," in *Proceedings of the 2011 ACM Symposium on Applied Computing*. ACM, 2011, pp. 1550–1557.

[18] B. Wang, H. Li, and M. Li, "Privacy-preserving public auditing for shared cloud data supporting group dynamics," in *Communications (ICC), 2013 IEEE International Conference on*. IEEE, 2013, pp. 1946–1950.

[19] M. Bellare and A. Palacio, "Gq and schnorr identification schemes: Proofs of security against impersonation under active and concurrent attacks," in *Advances in Cryptology– Crypto 2002*, vol. 2442. Santa Barbara, California, USA: Springer, August 18-22 2002, pp. 162–177.

[20] Y. Yu, J. Ni, M. H. Au, Y. Mu, B. Wang, and H. Li, "Comments on a public auditing mechanism for shared cloud data service," *IEEE Transactions on Services Computing*, vol. 8, no. 6, pp. 998–999, 2015.

[21] D. Chaum and T. P. Pedersen, "Wallet databases with observers," in *Advances in Cryptology– Crypto 1992*, vol. 92. Santa Barbara, California, USA: Springer, August 16-20 1992, pp. 89–105.

[22] Shamus Software Ltd., "Miracl library." http://www.shamus.ie/index.php?page=home.

[23] A. Juels and B. S. Kaliski Jr, "Pors: Proofs of retrievability for large files," in *Proceedings of the 14th ACM conference on Computer and communications security*. Acm, 2007, pp. 584–597.

[24] D. Boneh, B. Lynn, and H. Shacham, "Short signatures from the weil pairing," in *Advances in Cryptology– ASIACRYPT 2001*, vol. 2248. Gold Coast, Australia: Springer, December 9-13 2001, pp. 514–532.

[25] G. Ateniese, R. Di Pietro, L. V. Mancini, and G. Tsudik, "Scalable and efficient provable data possession," in *Proceedings of the 4th international conference on Security and privacy in communication netowrks*. ACM, 2008, p. 9.

[26] Q. Wang, C. Wang, J. Li, K. Ren, and W. Lou, "Enabling public verifiability and data dynamics for storage security in cloud computing," *Computer Security–ESORICS 2009*, pp. 355–370, 2009.

[27] C. C. Erway, A. Küpçü, C. Papamanthou, and R. Tamassia, "Dynamic provable data possession," *ACM Transactions on Information and System Security (TISSEC)*, vol. 17, no. 4, p. 15, 2015.

[28] J. Shen, J. Shen, X. Chen, X. Huang, and W. Susilo, "An efficient public auditing protocol with novel dynamic structure for cloud data," *IEEE Transactions on Information Forensics and Security*, 2017.

[29] H. Yan, J. Li, J. Han, and Y. Zhang, "A novel efficient remote data possession checking protocol in cloud storage," *IEEE Transactions on Information Forensics and Security*, vol. 12, no. 1, pp. 78–88, 2017.

[30] D. Cash, A. Küpçü, and D. Wichs, "Dynamic proofs of retrievability via oblivious ram," *Journal of Cryptology*, vol. 30, no. 1, pp. 22–57, 2017.