Multiple-Replica Public Auditing Protocol for Cloud Storage

Hongwei Liu¹, a, Shulan Wang², b, Peng Zhang³, c

¹ College of Information Engineering, Shenzhen University, Shenzhen, 518060, China
² College of Information Engineering, Shenzhen University, Shenzhen, 518060, China
³ College of Computer Science and Software Engineering, Shenzhen University, Shenzhen, 518060, China

aemail: liuhw@szu.edu.cn, bemail: wangshulan@email.szu.edu.cn, cemail: zhangp@szu.edu.cn

Keywords: Cloud Storage, Multiple Replica, Data Integrity, Public Auditing, Homomorphic Hash

Abstract. Cloud storage can provide a flexible on-demand data storage service to users anywhere and anytime. However, users’ data is owned by cloud service provider physically, and the physical boundary between two users’ data is fuzzy. Moreover, cloud storage provider stores multiple replicas of data in order to increase robustness of users’ data. The user is charged by the amount of replicas. However, the evidence cloud storage provider actually spends more storage space is scarce. In this environment, a method to ensure multi-replica integrity must be provided. In order to avoid retrieving enormous storage data and users themselves checking, a multi-replica public auditing protocol was proposed based on the BLS short signature scheme and the homomorphic hash function. Based on the computational Diffie-Hellman assumption, the presented protocol is secure against the lost attack and tamper attack from cloud service provider. As the independence among blocks and block signatures, this protocol supports block-level dynamic update, including insertion, modification and deletion. So, the protocol is secure and efficient, and supports for multi-replica, public verification, dynamic update and privacy preserving.

Introduction

As the excellent characteristics in terms of flexibility and scalability, cloud storage service, as the basic service of cloud computing, has been envisioned as the next generation of storage services. In cloud storage system, once the data is uploaded to the cloud without a copy in local computers, the user loses control of the data physically. So, cloud storage service also brings new and challenging security issues [1]. The cases cited in [2] also illustrate that cloud storage service suffers from internal or external data security threats in spite of the claimed completeness given by cloud service provider (CSP).

In order to check whether the data is tampered, deleted or lost, there are two kinds of ways to verify data integrity in cloud storage system: owner auditing [3] and public auditing [4][5][6][7]. With owner auditing, only users check the integrity of their remote stored data, which may introduce heavy overhead and cost. Avoiding any side of CSP or the data owner conducting the auditing, public auditing, transferring the auditing procedure to third party auditor (TPA), is a natural choice.

So as to increase robustness of users’ data, CSP may store multiple replicas of data. This practice is very common in cloud storage system. When some data blocks are tampered or lost, it’s easy to recover the corrupt data from the replicas. Of course, users must pay an additional fee for this special service, which is priced by the amount of replicas. So, in order to protect the interests of the users, not only the data integrity must be verified, but also the amount of replicas must be verified. Multiple-replica data integrity verification becomes a hotspot.

Wang et al. [4][5] firstly proposed the public auditing measures in cloud storage. In [4], they considered the task of allowing TPA, on behalf of the users, to verify the integrity of the dynamic data stored in the cloud servers. Remote data integrity was ensured with the support for both public verifiability and dynamic data operations, but users’ data privacy was not preserved. In [5], they also considered introducing TPA to audit the cloud data storage. They utilized public-key based
homomorphic authenticator and uniquely integrated it with random mask technique to achieve a privacy preserving public auditing system. However, the signature scheme used in [5] was insecure. In order to facilitate rapid deployment of cloud data storage service and regain security assurances with outsourced data dependability, Wang et al. [6] emphasized efficient methods that enabled on-demand data correctness verification on behalf of cloud data owners had to be designed. They described approaches and system requirements that should be brought into consideration for such a publicly auditable secure cloud storage service to become a reality. Zhu et al. [7] proposed a formal framework for interactive provable data possession (IPDP) and a zero-knowledge IPDP solution for private clouds. Their ZK-IPDP protocol achieved probabilistic data possession guarantee, supported fully data dynamics, public verifiability and was also private against the verifiers. Hao et al. [8] proposed a multiple-replica remote data possession checking protocol which had public verifiability. However, dynamic update and privacy preserving were not allowed. The above discussed protocols are compared, and the results are summarized in Table 1.

Table 1. Comparisons among the above discussed protocols

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Multiple replicas</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Public verifiability</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Dynamic Update</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Privacy preserving</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>

This paper aims at solving the security problem of the protocol in [5], and presenting a secure and efficient auditing protocol based on the homomorphism technology, which supports for multiple replicas, public verifiability, dynamic update and privacy preserving. The rest of this paper is organized as follows. In section II, a multiple-replica public auditing protocol is proposed based on the security flaw in [5]. We describe the support for dynamic update of the proposed protocol in section III. In section IV, the protocol’s complexity is analyzed in the aspects of communication, computation and storage costs, and experiments on the personal computer show that the protocol is feasible. Conclusions and possible future work are presented in section V.

The Multiple-Replica Public Auditing Protocol

To solve the security problem of the protocol in [5], the BLS short signature scheme proposed in [9] is used directly. Homomorphic hash function presented in [10] [11] is utilized to achieve the support for public auditing. Based on the system model, a multiple-replica public auditing protocol is described as following:

A. Setup

This section consists of three algorithms: KeyGen(1^λ) , ReplicaGen(F,s,k,n) and SigGen({C_1,…,C_n},sk).

KeyGen(1^λ). Input a security parameter λ . Choose a large prime p randomly. Define \( \mathbb{G}_1 \) and \( \mathbb{G}_2 \) to be multiplicative cyclic groups with order p. Let \( g \) be a generator of \( \mathbb{G}_1 \). \( H : \mathbb{Z}_p^* \rightarrow \mathbb{G}_1 \) is a homomorphic hash function defined in [16]. Mapping \( e : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_2 \) is a bilinear mapping. The user chooses \( x \leftarrow \mathbb{Z}_p^* \) randomly, and computes \( v \leftarrow g^x \in \mathbb{G}_1 \). The secret key for this user is \( sk = \{x\} \). The public keys for this user are \( pk = \{g,v\} \).

ReplicaGen(F,s,k,n). Let \( F \) be the user’s data file. Denote the number of the replicas by s. The user adopts a random key k to encrypt the data file F, and gets several different replicas \( C_i = E_k (i \| F) \), where \( 1 \leq i \leq s \). Accordingly, the authorized users can decrypt the data file \( F \) from CSP with the key k. As cloud storage service runs in a cooperated and distributed manner, to store each replica \( C_i \) in a set of cloud servers, \( C_i \) needs to be divided into n blocks \( c_{i,1}, \ldots, c_{i,n} \), where
\[ c_{i,j} \in \mathbb{Z}_p^*. \]

\[ \text{SigGen}(\{C_1, \ldots, C_s\}, sk). \] Given replicas \( \{C_1, \ldots, C_s\} \) and the secret key \( sk = \{x\} \), where each replica \( C_i = \{c_{i,1}, \ldots, c_{i,n}\} \). Define the identity of the data file \( F \) to be \( id \in \mathbb{Z}_p^* \). For each \( i \in \{1, \ldots, s\} \) and each \( j \in \{1, \ldots, n\} \), the user computes the signature

\[ \sigma_{i,j} \leftarrow (H(id \parallel i \parallel j)H(c_{i,j}))^x \in \mathbb{G}_1. \]

Then, the user sends the replicas \( C_i = \{c_{i,1}, \ldots, c_{i,n}\} \) and the corresponding signatures \( \{\sigma_{1,\ldots,\sigma_{i,n}}\} \) to CSP, where \( 1 \leq i \leq s \). Finally, the user deletes the replicas \( \{C_1, \ldots, C_s\} \) from local storage.

\section*{B. Challenge}

To verify that \( \{C_1, \ldots, C_s\} \) exist indeed in the cloud servers, the user sends an auditing request to TPA. Subsequently, TPA defines the subset of set \( \{1, s\} \) to be \( I = \{u_i\}_{1 \leq k \leq c} \) and \( u_1 \leq \ldots \leq u_c \). Meanwhile, TPA defines the subset of set \( \{1, n\} \) to be \( J = \{v_j\}_{1 \leq l \leq d} \) and \( v_1 \leq \ldots \leq v_d \). For each group \( (i \in I, j \in J) \), TPA chooses a random \( y_{i,j} \leftarrow \mathbb{Z}_p^* \), and generates \( \text{chal} = \{(i, j), y_{i,j}\}_{i \in I, j \in J} \) as the challenge. Then, TPA sends \( \text{chal} = \{(i, j), y_{i,j}\}_{i \in I, j \in J} \) to CSP. The blocks with positions \( (i, j) \) specified by \( \text{chal} \) are required to be checked.

\section*{C. Proof}

\[ \text{ProofGen}(\{c_{i,j}\}_{i \in I, j \in J}, \{\sigma_{i,j}\}_{i \in I, j \in J}, \text{chal}, pk) . \] Upon receiving \( \text{chal} = \{(i, j), y_{i,j}\}_{i \in I, j \in J} \), CSP chooses the blocks \( \{c_{i,j}\}_{i \in I, j \in J} \) and the signatures \( \{\sigma_{i,j}\}_{i \in I, j \in J} \) whose positions are \( I = \{u_1, \ldots, u_c\} \) and \( J = \{v_1, \ldots, v_d\} \).

If the linear combination of sampled blocks specified in \( \text{chal} \) can pass the verification from TPA, the blocks specified exist indeed in the cloud servers because of the randomness of the subset \( I = \{u_1, \ldots, u_c\} \) and \( J = \{v_1, \ldots, v_d\} \). So, one verification operation for the linear combination of blocks can replace multi verification operations for blocks. CSP computes the aggregate block

\[ \zeta = \sum_{i=1}^{u_c} \sum_{j=1}^{v_d} y_{i,j} c_{i,j} \quad \text{and the aggregate signature } \sigma = \prod_{i=1}^{u_c} \prod_{j=1}^{v_d} \sigma_{i,j}. \]

Next, CSP sends \( \text{Proof} = \{\sigma, \zeta\} \) to TPA.

\section*{D. Verification}

\[ \text{ProofVer}(\text{Proof}, pk) . \] Upon receiving \( \text{Proof} = \{\sigma, \zeta\} \) from CSP, TPA checks the verification equation:

\[ \hat{e}(\sigma, g) = \hat{e}(\prod_{i=1}^{u_c} \prod_{j=1}^{v_d} H(id \parallel i \parallel j)^{y_{i,j}} \cdot H(\zeta), v) \]

If the above equation holds, TPA returns TRUE, and CSP indeed processes the blocks \( \{c_{i,j}\}_{i \in I, j \in J} \). Otherwise, return FALSE.

In this protocol, \( I = \{u_1, \ldots, u_c\} \) is a subset of \( \{1, s\} \), and \( J = \{v_1, \ldots, v_d\} \) is a subset of \( \{1, n\} \). So \( |I| = c \leq s \) and \( |J| = d \leq n \). If \( c = s \) and \( d = n \), the verification for the replicas \( \{C_1, \ldots, C_s\} \) is determinate. If \( c < s \) or \( d < n \), the verification for the replicas \( \{C_1, \ldots, C_s\} \) is probabilistic. \( c, d \) can be defined according to the security level of the cloud storage system.

\section*{Dynamic Update}

In practical scenarios, the use may frequently perform block-level update operations on the data files (e.g. insertion, modification, deletion). During the dynamic update operations, most of update even a small portion of the file may require the user to update all block and signature indexes. To
mitigate this kind of drawbacks, the efficient dynamic update processing is advanced.

To simplify the description, we take one replica as the example, and assume the file blocks \( C_i = \{ c_{i,1}, \ldots, c_{i,n} \} \) and the signatures \( \{ \sigma_{i,1}, \ldots, \sigma_{i,n} \} \) have already been generated and properly stored at CSP. \( \text{Jump} \) is described as the next index number. If \( \text{Jump} = \text{NULL} \), the blocks are read sequentially. Otherwise, the next index number is the position specified by \( \text{Jump} \). To measure the communication overhead for data updates, the update communication factor \( \alpha \) is defined as the following:

\[
\alpha = \frac{\text{the row amount of data communication for updating one block}}{\text{the total row amount of data at CSP}}
\]

A. Block Insertion

Block insertion, a general form of data operation, refers to inserting a new block on the specified position in the data file \( C_i \). As described in Fig. 1, the user needs to insert \( c_{i,x} \) between \( c_{i,(j-1)} \) and \( c_{i,j} \). a) The user computes the signature \( \sigma_{i,x} \) of \( c_{i,x} \) by \( \text{SigGen}(c_{i,x}, sk) \), and sends \( \{ (i,j-1), (i,j), c_{i,x}, \sigma_{i,x} \} \) to CSP. b) Upon receiving them, CSP verifies the signature \( \sigma_{i,x} \), and then stores \( c_{i,x} \) and \( \sigma_{i,x} \) on \( (i, n+1) \). c) CSP modifies the jump of index number \( (i, j-1) \) to \( (i, n+1) \), the jump of index number \( (i, n+1) \) to \( (i, j) \). This processing involves updating the \( (i-1) \)-th row and appending the \( (n+1) \)-th row. Thus, \( \alpha = \frac{2}{n+1} \).

![Fig. 1. Inserting \( c_{i,x} \) between \( c_{i,(j-1)} \) and \( c_{i,j} \)](image)

B. Block Modification

Compared to block insertion, block modification does not change the logic structure of the data file \( C_i \). It refers to the replacement of specified blocks with new ones. As described in Fig. 2, the user needs to modify the \( (i, j) \)-th block \( c_{i,j} \) to \( c_{i,j}^* \). a) The user computes the signature \( \sigma_{i,j}^* \) of \( c_{i,j}^* \) by \( \text{SigGen}(c_{i,j}, sk) \), and sends \( \{ (i,j), c_{i,j}, \sigma_{i,j} \} \) to CSP. b) Upon receiving them, CSP verifies the signature \( \sigma_{i,j}^* \), and then replaces \( c_{i,j}, \sigma_{i,j} \) with \( c_{i,j}, \sigma_{i,j}^* \) respectively on \( (i, j) \). Thus, \( \alpha = \frac{1}{n} \).

![Fig. 2. Modifying \( c_{i,j} \) to \( c_{i,j}^* \)](image)
\( \sigma_{i,j} \) with NULL on \((i, j)\). c) CSP modifies the jump of index number \((i, j)\) to \((i, j+1)\). Thus, 
\[
\alpha = \frac{1}{n}.
\]

**Efficiency Analysis**

To validate the effectiveness and efficiency of our proposed protocol for public auditing, we simulate the public auditing service by using a personal computer with Intel Pentium processor at 2.93 GHz and 1.24GB RAM running Ubanut 10.10. The cryptographic library MIRACL is used. Let \(|p|=160\), \(m_j \in \mathbb{Z}_p^*\), \(c = n = 32\). Computation costs of the protocol in [5] and the proposed protocol are shown in Table 2.

Table 2. Computation Cost for Each Step of the Protocol in [5] and the Proposed Protocol with 32 Blocks

<table>
<thead>
<tr>
<th>Block</th>
<th>Signature</th>
<th>Jump</th>
<th>KeyGen (ms)</th>
<th>SigGen (ms)</th>
<th>ProofGen (ms)</th>
<th>ProofVer (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>The protocol in [5]</td>
<td>90</td>
<td>8</td>
<td>260</td>
<td>270</td>
<td>290</td>
<td></td>
</tr>
<tr>
<td>The proposed protocol</td>
<td>90</td>
<td>8</td>
<td>260</td>
<td>270</td>
<td>290</td>
<td></td>
</tr>
</tbody>
</table>

Let \(|p|=160\), \(m_j \in \mathbb{Z}_p^*\), \(c = n = 64\). Computation costs of the protocol in [5] and the proposed protocol are shown in Table 3.


<table>
<thead>
<tr>
<th>Block</th>
<th>Signature</th>
<th>Jump</th>
<th>KeyGen (ms)</th>
<th>SigGen (ms)</th>
<th>ProofGen (ms)</th>
<th>ProofVer (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>The protocol in [5]</td>
<td>110</td>
<td>8</td>
<td>520</td>
<td>540</td>
<td></td>
<td></td>
</tr>
<tr>
<td>The proposed protocol</td>
<td>110</td>
<td>8</td>
<td>510</td>
<td>530</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Assume the replicas \(C = \{c_{i,j},...c_{i,n}\}\), and the user needs to run the updating operation described in section V. The proposed dynamic update processing is to mitigate the communication overhead drawbacks in [12]. Define the position for update in [12] to be \(i\). Assume each replica is divided into \(n\) blocks originally. The update communication factor comparisons between the protocol in [12] and the proposed protocol are described in Table 4.

Table 4. The Update Communication Factor \(\alpha\) Comparisons between the Protocol in [12] and the Proposed Protocol.

<table>
<thead>
<tr>
<th>Block</th>
<th>Insertion</th>
<th>Modification</th>
<th>Deletion</th>
</tr>
</thead>
<tbody>
<tr>
<td>The protocol in [12]</td>
<td>(\frac{n-i+2}{n+1})</td>
<td>(\frac{1}{n})</td>
<td>(\frac{n-i}{n-1})</td>
</tr>
<tr>
<td>The proposed protocol</td>
<td>(\frac{2}{n+1})</td>
<td>(\frac{1}{n})</td>
<td>(\frac{1}{n})</td>
</tr>
</tbody>
</table>

Based on the dynamic update communication factor’s comparisons, the block insertion and deletion processing in the proposed protocol are more efficient than in [12].
Conclusion

To provide security cloud storage service to users, multi-replica public auditing system to verify data integrity must be considered. The multi-replica public auditing protocol was proposed in this paper. Without retrieving the data, we can check whether the data in cloud is lost or tampered by this proposed protocol, and there is no new computation cost for users. Efficiency analyses show the proposed protocol can solve the security flaw in [5] without more cost. Compared to the update processing in [12], this protocol is more efficient.

Acknowledgement

This research was supported by the National Natural Science Foundation of China under Grant No. 61001058 and No. 61171072, and the Key Program for Technology & Innovation of College in Guangdong Province No. CXZD1143.

References