Abstract
Cloud computing is the fastest growing and promising field in the sector of service provision. It is a very trendy situation for security implementation in the Cloud. Thus, security restrictions are increasing on users and service providers as the growth of Cloud environment. The purpose of this paper is to suggest a better and efficient integrity verification technique for data (referred as “Cloud Audit”). The building blocks of our technique are a variant of the Paillier homomorphic cryptography system with homomorphic tag and combinatorial batch codes. A Paillier Homomorphic Cryptography (PHC) system used to obtain homomorphic encryption on data blocks. Homomorphic tags along with the Paillier cryptography system assigns a special verifiable value to each data block, which helps us in unleashing data operations on this block. Combinatorial batch codes are used to assign and store integral data into different distributed cloud sever. To demonstrate our approach, we have implemented an application based on Hadoop and MapReduce framework. We have tested this application based on various parameters. Effectiveness of the proposed method has been shown by the experimental results. Our method has shown large improvement over the other modern methods.

Keywords: Combinatorial Batch Codes (CBC); Homomorphic Tag; Paillier Homomorphic Cryptography (PHC); Proof of Retrievability (PoR); Provable Data Possession (PDP); Third Party Auditing.

1. Introduction
Cloud Computing (CC)\textsuperscript{1,2} is an on-demand service over a network servers which are hosted on Internet to process, store and organize the data, rather than a local server or personal computer. The Cloud services and applications runs on distributed network which provides a virtual resource for end user. This resources could be accessed by standard Internet and networking protocols. Data integrity verification\textsuperscript{3} is one of the massive responsibility with cloud data, because the probability of involvement in malicious activity of a cloud user and cloud provider is very high. There are many way to address this problem. User can use encryption and decryption process. However, it requires huge computing time and functional overheads. Applying data auditing may be the other way to address this problem.

Classical approaches for data auditing are Provable Data Possession (PDP) techniques\textsuperscript{4–7,8} and Proof of Retrievability (PoR) techniques\textsuperscript{9–13}. However, there are many difficulties with PDP and PoR techniques. These techniques can be employed only for encrypted files and allow only a limited number of queries. Further, the above
schemes are not suitable for the batch auditing because of their computation overhead. The benefits of these techniques only depend on the preprocessing steps which user can apply on outsource data file. There is a settlement between dynamic data operations and privacy preservation. But, some of the schemes do not preserve privacy. There is a trade-off between cost of communication and storage overhead. However, some of these PDP schemes require high cost for less storage.

There are some schemes\textsuperscript{14–17} which assign audit tasks to single Third Party Auditor (TPA). TPA is an authentic and trusted entity that independently manages the data audits.

To address the above problems, Saxena \textit{et al.}\textsuperscript{3} have proposed multiple TPA scheme, in which various synchronous audit sessions from different users are handled by each single TPA simultaneously. We assume for our scheme that cloud service provider must be trustworthy.

Major contributions of our approach are as follows:

1. We provide public audit structure with multiple TPA for privacy preserve audit and data storage security.
2. Our approach supports dynamic data operations and provides an effective solution for public auditing with multiple TPA.
3. Proposed scheme can handle various efficient and synchronous audit sessions from multiple users.
4. Effectiveness of the proposed scheme is demonstrated through implementations and experimentations.

**Organization.** We have organized the remaining paper in three sections. Section 2 will discuss our proposed scheme for data integrity verification. In section 3, we describe the support of dynamic data operations. Security and performance analysis are given in Section 4. Finally, in section 5, we presents the conclusions of our work.

2. Proposed Scheme

In this section, first, we present the notation related to our approach. Then we describe the details of Paillier Homomorphic Cryptography system (PHC)\textsuperscript{18}. Subsequently, we illustrate the system model and details of our scheme.

2.1 Notation and preliminaries

1. $F$ and $F[i]$: a file and the $i^{th}$ data block of $F$.
2. $E(\bullet)$ and $D(\bullet)$: the encryption and decryption algorithms.
3. $H(\bullet)$: a hash function.
4. $\phi(\bullet)$: Euler’s totient function.
5. $\mu(\bullet)$: a Pseudo-Random Function (PRF) which maps: $\mu: \{0, 1\}^k \times \{0, 1\}^l \rightarrow \{0, 1\}$.
6. $\sigma(\bullet)$: a Pseudo-Random Permutation (PRP) which maps: $\sigma: \{0, 1\}^k \times \{0, 1 \ldots n\} \rightarrow \{0, 1 \ldots n\}$.
7. $p$ and $q$: two different odd prime numbers of the same length.
9. $r_1, r_2$: random numbers selected from the Galois field.
10. $T$ and $T_i$: all block tags and $i^{th}$ tag of $T$.
11. $v$: number of verification.
12. $R$: number of blocks required for each challenge.
13. $k_t$: encryption key for tag.
14. $k_d$: decryption key for tag.
15. $Z$ and $Z^*_N$: group on integer numbers.
16. $c$: number of blocks that choose for challenge operation.
17. $r$: number of deleted blocks from total file blocks.

2.2 Paillier homomorphic cryptography system

We use a variant of PHC system\textsuperscript{18} for encryption and decryption. Group of integer numbers ($Z_N$ and $Z^*_N$) are utilized such that $Z_N \times Z^*_N$ is isomorphic to $Z^{12}_N$. PHC system has three parts which are described below.
2.2.1 Key-generation

In first part, PHC system generates public and private keys for encryption and decryption, respectively a random number which is used to encrypt plain-text. It applies Euler’s totient function on two different odd prime numbers to generate these keys. The procedure of key-generation is summarized as follows.

1. An entity chooses two different odd prime numbers \( p \) and \( q \) of the same length.
2. Calculate \( J = pq \) and Euler’s totient function on \( J \) is \( \phi(J) = ((p-1)(q-1)) \).
3. Assure that
   
   a) \( \gcd(J, \phi(J)) = 1 \).
   
   b) For any integer \( a > 0 \), we have \( (1 + J)^a = (1 + aJ) \mod J^2 \).
   
   c) As a consequence, the order of \( (1 + J) \in \mathbb{Z}_J^* \) is \( \phi(J) \) i.e. \( (1 + J)^J = (1 \mod J^2) \) and \( (1 + J)^a \neq (1 \mod J^2) \) for any \( 1 < a < J \).
4. Selects a random \( r \in \mathbb{Z}_J^* \) such that \( \gcd(L(rJ \mod J^2), J) = 1 \), where \( L(x) = (x - 1)/J \).
5. Return public key \( (J) \), private key \( (J, \phi(J)) \) and a random number \( (r) \) of the system.

2.2.2 Encryption

Second part of PHC system encrypts plain text using public key. Let \( m \in \mathbb{Z}_J \) be a plain-text to be encrypted and \( r \in \mathbb{Z}_J^* \) is a random number. With the definition of isomorphism, cipher-text can be obtained by function \( f \) that maps plain text as:

\[
\mathbb{Z}_J \times \mathbb{Z}_J^* \rightarrow \mathbb{Z}_J^2 \quad \text{and} \\
\hat{c} = E(m \mod J, r \mod J) = f(m, r) = [(1 + J)^m \cdot r \mod J^2]; \quad \text{where} \quad c \in \mathbb{Z}_J^2.
\]

2.2.3 Decryption algorithm

In the last part of PHC system, user can efficiently decrypts the encrypted text using its private key \( (J, \phi(J)) \). Decryption steps are given as follows.

1. Set \( \hat{c} := [c \phi(n) \mod J^2] \) where \( c \) is cipher-text.
2. Set \( \hat{m} := (\hat{c} - 1)/J \). (Note that all this is carried out over the integers.)
3. After decryption, plain-text is given by \( m := [\hat{m} \cdot \phi(J)^{-1} \mod J^2] \)

2.3 Cloud audit: proposed data integrity verification scheme

In this paper, we propose multiple and distributed third party (Cloud Audit), which shares the massive load of batch auditing among multiple TPAs by load balancing techniques. Our scheme works on three tiers which are given in the following.

1. Cloud Users: Any consumer of cloud service could be represented as a cloud user. Our assumption for cloud user is that they have limited resources and are not capable of performing computation intensive jobs, such as different auditing tasks (privacy preserving data integrity verification).
2. Multiple TPA: TPA is an authentic and authorize entity. Data integrity verification and batch auditing tasks are managed by TPAs. Further, it is also responsible for Service Level Agreement (SLA) and different legal issues of cloud providers. To accomplish the load balance task and perform batch audit, we use multiple TPAs.
3. Cloud Service Provider (CSP): CSPs are the organizations who provide sufficient infrastructure and resources to as a service to the end users. Distribution of resources could be within many servers, which are situated on around the world.
Our proposed Cloud Audit technique works in two phases: startup and audit phases. Figure 1 and Fig. 2 describe the working of startup and audit phases within the three tiers, respectively.

A. Startup Phase: The startup performs four tasks which are described in the following.

1. Request for Data Storage: In the first step of startup phase, cloud user sends a request for storing the data of a file $F$ to the CSP group.

2. Storing the File $F$: At CSP end, file $F$ is converted into $n$ data blocks by file partitioning operation. To maintain high availability, CSP makes replica of each block two times. We assume that CSP group uses Combinatorial Batch Code (CBC) $\mathcal{C}(n, N, k, m, t)$ to store the file $F$. Thus, a file of $n$ data blocks is to be stored among $m$ servers in a specific way that any $k$ of the $n$ data blocks can be recovered by at max $t$ blocks from each server, and the total blocks stored in $m$ servers are $N$. Initially, CSP group performs setup operation and generates public encryption key of a TPA group and two random numbers. Steps of this task are summarized in Algorithm 1.

Algorithm 1. Setup Operation

| Input: $\{0, 1\}^p$. |
| Output: Private decryption key ($k_0$) of TPA group, public encryption key ($k_i$) of TPA group, Random numbers $r_1$, $r_2$. |

1. CSP chooses $p$ and $q$
2. Call Key-Generation Algorithm of PHC system
3. PHC system generates $(J)$ and $(\psi(J))$ as public and private keys, respectively.
4. Assign $(J)$ as public key i.e. $k_i = (J)$ and $(\psi(J))$ as private key i.e. $k_i = (J, \psi(J))$ of TPA group.
5. Two random numbers $r_1$ and $r_2$ generated by $r_1 \leftarrow \{0, 1\}^p$ and $r_2 \leftarrow \{0, 1\}^p$, respectively.

3. Homomorphic TAG Generation: In the second step, CSP group performs a homomorphic tag generation process. In this process, each CSP server individually chooses random sample blocks $F_1, F_2, \ldots, F_k$ from total $n$ data blocks and compute hash value using $H(\cdot)$ for each block. These hash values are represented as $H(F_1), H(F_2), \ldots, H(F_k)$. Load balancing mechanism is applied to distribute this process among all CSP servers. Let $\nu$ be the number of data blocks is assigned to each CSP server after load balancing. Hence, each CSP server will generate homomorphic tag for those $\nu$ number of data blocks. Tag generation process is described in Algorithm 2.
4. **Return Encrypted Tag** $T$: CSP group returns encrypted tag $(T)$ entry, which contains tags $T_i$ for all data blocks of a cloud user. Although, this tag increases the size of metadata related to file, yet this tag helps cloud users in performing dynamic operations on their data without disclosing the content to anyone. This tag also helps TPA group for data integrity verification, which will be described in auditing phase.

**B. Auditing Phase:** The auditing phase verifies the data integrity by performing five tasks which are described in the following.

1. **Request for Data Integrity Verification:** The cloud user uses the encrypted entry and file information to check the data integrity. TPA group is responsible to audit the data integrity on behalf users. Hence, user requests the TPA group for auditing of a file $F$ by sending the message $\{F, T\}$ to TPA. TPA group performs the audit task by sharing the audit load among other TPAs by load balancing method.

2. **Challenge:** In challenge operation, TPA group may challenge CSP for some random data blocks ($c$). For this purpose, TPA group computes $c_i$ for the $i^{th}$ challenged block using the public key ($k_r$) of CSP. Then a message $(r_2, c_i)$ is sent to the CSP server. Details of challenge operation is given in Algorithm 3.

3. **Proof Generation:** In this step, CSP server decrypts the challenge block entry using its private key. Then, it finds the positions of the requested challenged data blocks using $c_i$. Finally, sum of the retrieved data blocks $(F'_i)$ are calculated and then send the sum $(F'_i)$ to the TPA group. CSP server also returns the homomorphic verifiable tag $(T'_i)$ corresponding to $F'_i$. The tag, $T'_i$, is already stored on CSP. The procedure for proof generation is described in Algorithm 4.
Algorithm 4. Proof Generation

4. **Proof Verification:** This process verifies the homomorphic tags of challenged blocks. To do this, $T'_i$ is decrypted and the Hash of $F'_i$ is calculated by TPA group. For decryption, TPA group uses their private key $k_d$. The decrypted tag is compared with generated hash value to check whether they are equal or not using proof verification process.

If the result of verification process is yes, then it indicates that the integrity of the data blocks (file) is preserved, else the data blocks (file) are manipulated. Summary of the proof verification process is given in Algorithm 5.

Algorithm 5. Proof Verification

```
Input: D(•), $k_d$ of TPA group.
Output: Verification result $H(F'_i) \equiv \rho_i$.
1: $\rho_i = D_{k_d}(T'_i)$.
2: Verifies $H(F'_i) \equiv \rho_i$.
```

Algorithm 5. Proof Verification

3. **Support for Dynamic Data Operations**

3.1 **Update operation**

In some situation, user modifies the existing file blocks and this modification restructures the whole file. This operation is known as update operation. To implement the update operation, we use the first homomorphic property of a Paillier cryptography system. This property is described with our notation as follows:

Let $m_1, m_2$ be any two file blocks containing plain-texts such that $m_1, m_2 \in \mathbb{Z}_J$ and $r_1, r_2$ be two random numbers such that $r_1, r_2 \in \mathbb{Z}_J^*$. First homomorphic property says that the decryption of the product of two cipher-texts is equal to the sum of their corresponding plain-texts. This could be represented by the following equation

$$D[E(m_1, r_1) \cdot E(m_2, r_2)] = D[E(m_1 + m_2, r_1, r_2) \mod J^2] = m_1 + m_2 \mod J$$

Thus, we are able to get the dynamic update operation and the addition of two plain-text without retrieving the plain-texts.

3.2 **Append operation**

Sometime, user may add new data block at the end of the stored files which causes increment in the size of the stored data. This operation is referred as append operation. To implement this operation, we use the second homomorphic property of a Paillier cryptography system, which is described with our notation as follows:

For any $m_1, m_2 \in \mathbb{Z}_J$ and $r_1, r_2 \in \mathbb{Z}_J^*$, decryption of an encrypted plain-text raised to the power of another plain-text is equal to the product of two plain-texts. This could be expressed by the following equations

$$D[E^{m_2}(m_1, r_1)] = D[E(m_1 \cdot m_2, r_1^{m_2}) \mod J^2] = m_1 \cdot m_2 \mod J$$
$$D[E^{m_1}(m_2, r_2)] = D[E(m_1 \cdot m_2, r_2^{m_1}) \mod J^2] = m_1 \cdot m_2 \mod J$$

Thus, we could ensure the dynamic appending of data and multiplication of two plain-text without retrieving plain-texts.
3.3 Delete operation

In some occasions, after saving the data on the cloud, user needs to delete some data blocks. We consider this operation as delete operation. This can be considered as a specialized update operation because we can replace the original data blocks with null blocks or any special predetermined symbol blocks.

4. Analysis

This section, first, presents the security analysis of our proposed method using binding property of the Paillier cryptography system. Further, we have discussed the experimental results to judge the performance of our scheme.

4.1 Security analysis

The goal of our technique is to ensure that solely authorized users can read, use, or contribute to the data stored at CSP. These security controls add another layer of stability against possible threats by cloud users, administrators and other vulnerable actors on the network. To ensure better security, we use the self binding property of Paillier cryptography system, which is described with our notation as follows:

\[ [E(m_1, r_1) \cdot r_2^f] \mod J^2 = E(m_1, r_1 \cdot r_2) \]

With this property, any cipher-text can be changed to another cipher-text without affecting the plain-text. We utilize this property for making the job of adversary very difficult to predict the plain-text.
Table 1. Experimental Setup.

<table>
<thead>
<tr>
<th>No. of PCs</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processor</td>
<td>Intel i7-2600S 2.80 GHz</td>
</tr>
<tr>
<td>Memory</td>
<td>16 GB</td>
</tr>
<tr>
<td>File Storage Server</td>
<td>Citrix Xen Server 6.2.0</td>
</tr>
<tr>
<td>Cloud Environment</td>
<td>Cloudera CDH 5.3.0-G</td>
</tr>
<tr>
<td>Maximum Data for Storage</td>
<td>1TB</td>
</tr>
</tbody>
</table>

Fig. 3. $\rho_{qt}$ for Different Values of $P_x$.

Fig. 4. $\rho_{qt}$ for Different Values of $F_s$.

4.2 Performance analysis

In our scheme, we can set number of queried data blocks and number of challenged data blocks according to user’s requirement. When users wish to store data for a short period, users can use less number of challenge blocks to reduce the overload of CSP and TPA.

We have executed our experiment with the setup given in Table 1. We have configured a TPA group in cloud environment to perform audit task of the stored files on the behalf of cloud users.

To measure the performance of the proposed scheme, the ratio of queried data blocks and the total number of data blocks is computed varying three parameters: detection probability ($P_x$), file size ($F_s$) and audit frequency ($A_f$). The ratio is defined as $\rho_{qt}$ We have also compared our approach with existing approaches4, 8, 9, 14, 16. The experimental results are given in the following.
1. **Detection Probability** ($P_X$): We have computed $\rho_{qt}$ for the different values of $P_X$ (0.75, 0.80, 0.85 and 0.90). The experimental results of our approach and existing approaches are given in Fig. 3. We can see that for any value of $P_X$ our scheme have a less $\rho_{qt}$.

2. **File Size** ($F_s$): We have tested our method with different file size ($F_s = 128$ MB, 256 MB, 512 MB and 1 TB) for audit purpose and measured $\rho_{qt}$. Figure 4 shows the results of our method as well as existing approaches with different values $F_s$. It may be observed that for any value of $F_s$ our scheme produce less $\rho_{qt}$.

3. **Audit Frequency** ($Af$): In our experiment, we have considered different audit frequency values ($Af = 80$ Hz, 85 Hz, 90 Hz and 95 Hz) to evaluate the proposed scheme and compare with existing approaches. Experimental results are presented in Fig. 4 which shows that our approach requires less $\rho_{qt}$ than all existing approaches for all $Af$ values.

5. **Conclusions**

In this paper, we propose a better data integrity verification approach which use multiple third party auditors. This approach uses CBC, PHC and homomorphic tag for data integrity verification. Ideally, the approach is suitable for cloud storage because of the efficiency of homomorphic tag and advantages of PHC. Further, our technique supports dynamic data operations with less overhead. In this approach, CSP server does not require any additional data structure to manage data operations. It also provides better security in case of Man In The Middle attack (MITM), Traffic flow analysis, Impersonation, Defacement and misuse of data storage servers, because of the self binding property of Paillier which can change cipher-text without any modification in plain-text and misguide the intruders. Finally, performance of our approach is not bounded with disk I/O and comparison with existing approaches shows the effectiveness and usefulness of our approach. Currently, the scheme is tested with limited data size (1TB) and with limited audit frequency (Max 95 Hz). In future, our approach could be extended for large data size and higher audit frequency.

**References**


