An Efficient and Secure Protocol for Ensuring Data Storage Security in Cloud Computing

Syam Kumar P, Subramanian R
Department of Computer Science, School of Engineering & Technology
Pondicherry University, Puducherry-605014, India

Abstract
Currently, there has been an increasing trend in outsourcing data to remote cloud, where the people outsource their data at Cloud Service Provider(CSP) who offers huge storage space with low cost. Thus users can reduce the maintenance and burden of local data storage. Meanwhile, once data goes into cloud they lose control of their data, which inevitably brings new security risks toward integrity and confidentiality. Hence, efficient and effective methods are needed to ensure the data integrity and confidentiality of outsource data on untrusted cloud servers. The previously proposed protocols fail to provide strong security assurance to the users. In this paper, we propose an efficient and secure protocol to address these issues. Our design is based on Elliptic Curve Cryptography and Sobol Sequence (random sampling). Our method allows third party auditor to periodically verify the data integrity stored at CSP without retrieving original data. It generates probabilistic proofs of integrity by challenging random sets of blocks from the server, which drastically reduces the communication and I/O costs. The challenge-response protocol transmits a small, constant amount of data, which minimizes network communication. Most importantly, our protocol is confidential: it never reveals the data contents to the malicious parties. The proposed scheme also considers the dynamic data operations at block level while maintaining the same security assurance. Our solution removes the burden of verification from the user, alleviates both the user's and storage service's fear about data leakage and data corruptions. Through security analysis, we prove that our method is secure and through performance and experimental results, we also prove that our method is efficient. To compare with existing schemes, our scheme is more secure and efficient.

Keywords: *data storage, integrity, confidentiality, Elliptive Curve Cryptography(ECC), Sobol Sequence, Cloud Computing.*

1. Introduction

Cloud storage becomes an increasing attraction in cloud computing paradigm, which enables users to store their data and access them wherever and whenever they need using any device in a pay-as-you-go manner[1]. Moving data into cloud offers great conveniences to the users since they do not have to care about the large capital investment in both the maintenance and management of the hardware infrastructures. Amazon's Elastic Compute Cloud (EC^2) and Amazon Simple Storage Service (S3) [2] and apple icloud[3] are well known examples of cloud data storage. However, once data goes into cloud, the users lose the control over the data. This lack of control raises new formidable and challenging issues related to confidentiality and integrity of data stored in cloud [4].

The confidentiality and integrity of the outsourced data in clouds are of paramount importance for their functionality. The reasons are listed as follows [5]:
1) the CSP, whose purpose is mainly to make a profit and maintains a reputation, has intentionally hide data loss an incident which is rarely accessed by the user's 2) The malicious CSP might delete some of data or is able to easily obtain all the information and sell it to the biggest rival of Company. 3) An attacker who intercepts and captures the communications is able to know the user's sensitive information as well as some important business secrets. 4) Cloud infrastructures are subject to wide range of internal and external threats.
The examples of security breaches of cloud service providers appear from time to time [6, 7]. The users require that their data remain secure over the CSP and they need to have a strong assurance from the cloud servers that CSP store their data correctly without tampering or partially deleting because the internal operation details of service providers may not be known to the cloud users. Thus, an efficient and secure scheme for cloud data storage has to be in a position to ensure the data integrity and confidentiality.

Encrypting the data before storing in cloud can handle the confidentiality issue. However, verifying integrity of data is a difficult task without having a local copy of data or retrieving it from the server. Due to this reason, the straightforward cryptographic primitives cannot be applied directly for protecting outsourced data. Besides, a naive way to check the data integrity of data storage is to download the stored data in order to validate its integrity, which is impractical for excessive I/O cost, high communication overhead across the network and limited computing capability. Therefore, efficient and effective mechanisms are needed to protect the confidentiality and integrity of user's data with minimum computation, communication and storage overhead.

Remote data integrity checking is a protocol that focuses on how frequently and efficiently we verify whether cloud server can faithfully store the user's data without retrieving it. In this protocol, the user generates some metadata. Later, he can challenge the server for integrity of certain file blocks through challenge-response protocol. Then the server generates responses that the server still possesses the data in its original form to corresponding challenge sent by the verifier who may be original user or trusted third party entity. Recently, several researchers have proposed different variations of remote data integrity checking protocols under different cryptography schemes [8-21]. However, all these protocols focus on static data verification.

One of the design principles of cloud storage is to provide dynamic scalability of data for various applications. This means, the data stored in cloud are not only accessed by the users but also frequently updated through block operations such as modification, insert and delete operations. Hence, it is crucial to develop more secure and efficient mechanism to support dynamic audit services. The protocols to verify dynamic data in cloud are proposed in [22-27].

Although the existing schemes aim at providing integrity verification for different data storage systems, but problem of confidentiality of data has not been fully addressed.

The protocols [28-35] have been proposed to ensure the confidentiality and integrity of remote data. But, all these schemes are unable to provide strong security assurance to the users, because these schemes verifying integrity of outsourced data based on pseudorandom sequence, which does not cover the whole data while computing the integrity proof. Therefore, probabilistic verification schemes based on pseudorandom sequence does not give guarantee to the users about security of their data. Syam et al. [27] proposed a distributed verification protocol using Sobol sequence to ensure availability and integrity of data, but it is also not addressed the data confidentiality issue. How to achieve a secure and efficient design to seamlessly integrate these two important components for data storage service remains an open challenging task in Cloud Computing.

In this paper, we propose an efficient and secure protocol to ensure the confidentiality and integrity of data storage in cloud computing using Elliptic Curve Cryptography(ECC) [30, 37, 38] and Sobol Sequence [39]. The ECC can offer same levels of security with small keys comparable to RSA and other PKC methods. It is designed for devices with limited computing power and/or memory, such as smartcards, mobile devices and PDAs. In our design, first the user encrypts data to ensure the confidentiality, then, compute metadata over encrypted data. Later, the verifier can use remote data integrity checking protocol to verify the integrity. The verifier should able to detect any changes on data stored in cloud. The security of our scheme relies on the hardness of specific problems in Elliptic Curve Cryptography. Compared to existing schemes, our scheme has several advantages: 1) it should detect all data corruption if anybody deletes or modifies the data in cloud storage, since we are using Sobol sequence instead of pseudorandom sequence for challenging the server for the integrity verification. 2) Our scheme achieves the confidentiality of data 3) It is efficient in terms of computation, storage, because its key size is low compared to RSA based solutions.

Main Contributions:

1) We propose an efficient and secure protocol. This protocol efficiently provides the integrity assurance to the users with strong evidence that the CSP is in faithfully storing all data and this data cannot be leaked to malicious parties. Our protocol also supports public verifiability and dynamic data operations such as modification, insertion and deletion.
2) We prove the security (integrity and confidentiality) of proposed scheme against internal and external attacks. Cloud server can provide valid response to the verifier challenges only if they actually have all data in an uncorrupted and update state.
3) We justify the performance of proposed protocol through concrete analysis, experimental results and comparison with existing schemes.

The rest of paper is organized as follows: **Section 2** describes the related works. **Sections 3** introduce the system model: including: cloud storage model, security threats, design goals and notations and permutations. In **Section 4,** we provide the detailed description of our scheme**.** **Section 5** gives the security analysis and **Section 6** gives the performance and experimental results and in **Section 7** we give conclusion to our work.

2. Related Work

The security of remote storage applications has been increasingly addressed in the recent years, which has resulted in various approaches to the design of storage verification primitives. The literature distinguishes two main categories of verification schemes [30]: Deterministic verification schemes check the conservation of a remote data in a single, although potentially more expensive operation and probabilistic verification schemes rely on the random checking of portions of outsourced data.

2.1. Deterministic Secure Storage

Deterministic solutions are verifying the storage of the entire data at each server. Deswarte et al. [8] and Filho et al.[9] are firstly proposed a solution to remote data integrity. Both use RSA-based functions to hash the whole data file for every verification challenge. They require pre-computed results of challenges to be stored at verifier, where a challenge corresponds to the hashing of the data concatenated with a random number. However, both of them are inefficient for the large data files, which need more time to compute and transfer their hash values. Carmoni et al. [10] described a simple deterministic approach with unlimited number of challenges is proposed, where the verifier like the server is storing the data. In this approach, the server has to send MAC of data as the response to the challenge message. The verifier sends a fresh unique random value as the key for the message authentication code to prevent the server from storing only the result of the previous hashing of the data. Golle et al. [11] proposed a SEC (Storage Enforcing Commitment) deterministic verification approach. This approach uses homomorpic verifiable tags, whose number is equal to two times of number of data chunks and the verifier choose a random value that will be used to shift indexes of tags to be associated with the data chunks when the integrity proof constructed by the server. Sebe et al. [12] presented a remote data checking protocol such that it allows an unlimited number of verifications and the maximum running time can be chosen at setup time and traded off against storage at verifier. However, none of the schemes were considered the problem of remote data confidentiality and dynamic data verifications.

To ensure the confidentiality of remote data, Shah et al. [27, 28] proposed a privacy-preserving audit protocol, which allows a third party auditor to keep online storage honest. In their schemes, the client first encrypts the data file and pre-computes a hash value over encrypted data using keyed hash function and sends it to the auditor. But, their schemes may potentially bring on-line burden to the users when the keyed hashes are used up. Oualha et al. [30] described a secure protocol for ensuring self organizing data storage (P2P) through periodic verifications, these verifications used for the integrity checks since each holder generates a response that they still having the data safely. In particular a data owner can prevent data damage at a specific holder by storing encrypted replicas crafted the use of elliptic curve cryptography. Wang et al. [31] proposed a privacy-preserving public auditing scheme for data storage security in cloud computing by using homomorphic authenticator and random masking. This scheme conceals the content of the original data from the TPA but not from the malicious servers. Similarly, Hao et al. [32] introduced the multiple replicas remote data possession checking protocol with public verifiability. However, this scheme does not support to dynamic data operations. In their subsequent work, Hao *et al.* [33] proposed a RSA-based privacy-preserving data integrity checking protocol with data dynamics in cloud computing. Their scheme extended the sebe's protocol [12] to support public verifiability. It does not leak any information to third party auditors. However, like[31] it is also not protecting data leakage from the malicious servers.

2.2. Probabilistically Secure Storage

The probabilistic verification schemes verify the specific portions of data instead of entite data at servers.

Ateniese et al. [13] proposed a RDC using PDP. In their system, the client pre-computes the tags for each block of a file using homomorphic verifiable tags and stores the file and it tags with the server. Then, the client can verify that server integrity of the file by generating a random challenge, which specifies the selected positions of file blocks. Using the queried blocks and their corresponding tags and the server generates a proof of integrity. Juels et al. [14] proposed a formal definition of POR and its security model. In this model, the encrypted data is being divided into small data blocks, which are encoded with Reed-Solomon codes. The "sentinels" are embedded among encrypted data blocks to detect whether it is intact. However, this can verify only limited number of times because this scheme has only finite number of "sentinels" in the file. When the finite "sentinels" are exhausted, the file must be sent back to the owner to re-compute new "sentinels". Ateniese et al. [15] proposed a new scheme with homomorphic linear authenticators (HLA) of which communication complexity is independent of the file length. This scheme supports unlimited number of verification, but it cannot verify publicly. Later, Shacham et al. [16] proposed the two POR protocols: The first one built from BLS signatures and has the shortest query and response with public verifiability. The second one is based on pseudorandom functions (PRFs) with private verifiability, but it requires a longer query. Both schemes rely on the homomorphic property-aggregating verification proofs into a small value. Dodis et al [17] first formally define the POR code, this construction improves the prior POR constructions. The main insight of their work comes from the simple connection between POR schemes and the notion of hardness amplification, extensively studied in complexity theory. Browers et al. [18] introduced a theoretical framework for previous POR protocols [14-16] using integrated forward error-correcting codes. In their subsequent work, Browers et al. [19] described a HAIL (High-Availability and Integrity Layer), in which the key insight is to embedded MACs in the parity blocks of the dispersal code. As both MACs and parity blocks can be based on universal hash functions. Schwarz et al. [20] used a XOR-based, parity m/n erasure codes to create *n* shares of a file that stored at multiple sites. Curtomola et al. [21] extended the PDP [13] to the multiple servers, which are called Multiple Replica-Provable Data Possession (MR-PDP), it is aimed to ensure availability and reliability of data across distributed servers. In this scheme, the user stores multiple replicas of a single file across distributed servers, thus we can get an original file from any one of the servers even if any server fails.

Although all these schemes are aim at providing integrity verification for different data storage systems but the problem of data dynamics has not at fully addressed.

For the dynamic data integrity verification, Ateniese *et al.* [22] have designed a highly efficient and provably secure PDP with data dynamics is called "Scalable Data Possession". It was based on symmetric key cryptography, while not requiring any bulk encryption. It improves the RDC[13] in terms of storage, bandwidth and computation overheads. However, it cannot perform block insertions anywhere beacause each update requires re-computing the all the remaining tokens, which is problematic for large files. In addition, it does not support public verifiability. Similarly, Wang et al. [23] discussed the problem of ensuring the availability and integrity of data storage in cloud computing. They utilized the homomorphic token and error correcting codes to achieve the integration of storage correctness insurance and data error localization, but like[22] their scheme do not support an efficent insert operation due to the index positions of data blocks. To overcome this probem, Erway et al. [24] firstly proposed a scheme to support dynamic data operations effieciently at block level instead of index positions[22, 23] by using rank-based verification skip list in the cloud servers. Later, Wang *et al.* [25] described a BLS based homomorpic authenticator with public verifiability and supports of data dynamics using Merkle Hash Tree (MHT) to verify the data integrity checking in cloud computing. They achieved the data integrity assurance with high efficiency. Similarly, Zhu *et al.* [26] proposed a dynamic auditing service for verification of integrity of outsourced data in cloud. Their design is based on fragment structure, random sampling and index-hash table. Their scheme achieved the integrity assurance with low computation, storage and computation overhead.

However, none of the schemes were address the problem of outsorced data confidentiality.

Ayad et al. [34] proposed a Provable Possession and Replication of Data over Cloud Servers with dynamic data support. This scheme achieves the availability, integrity and confidentiality of data storage in cloud. Chen et al. [35] described an efficient remote data possession in cloud computing. It has several advantages while achieving security of remote data as follows: First, it is efficient in terms of computation and communication. Second, it allows verification without the need for the challenger to compare against the original data. Third, it uses only small challenges and responses, and users need to store only two secret keys and several random numbers. Yang et al. [36] proposed a Provable Data Possession of Resource-constrained Mobile Devices in Cloud Computing. In this framework, the mobile terminal devices only need to generate some secret keys and random numbers with the help of trusted platform model (TPM) chips, and the needed computing workload and storage space is fit for mobile devices. Like [25], by using bilinear signature and Merkle hash tree (MHT), this scheme aggregates the verification tokens of the data file into one small signature to reduce communication and storage burden.

All these schemes are unable to provide strong security assurance to the users because all these schemes are verifying integrity of data using pseudorandom sequence. It does not cover the whole data while computing integrity proof. Therefore, probabilistic verification schemes based on pseudorandom sequence does not give strong guarantee to the users about security of their data.

To overcome this problem, Syam *et al.* [27] proposed a homomorpic distributed verification protocol to ensure data storage security in cloud computing using Sobol Sequence instead of pseudorandom sequence, which is more uniform than pseudorandom sequence. Their scheme achieves the availability and integrity of outsourced data in cloud but similar [23], it is also not addressing data confidentiality issue.

To achieve all these security and performance requirements of cloud storage, we propose an efficient and secure protocol in section 4.

3. System Model

3.1. Cloud Data Storage Model

The cloud storage model considering here is consists of three main components as illustrated in Fig. 1.

1) **Cloud User:** the user, who can be an individual or an organization originally storing their data in cloud and accessing the data.

2) **Cloud Service Provider (CSP):** the CSP, who manages cloud servers (CSs) and provides a paid storage space on its infrastructure to users as a service.

3) **Third Party Auditor (TPA) or Verifier:** the TPA or Verifier, who has expertise and capabilities that users may not have and verifies the integrity of outsourced data in cloud on behalf of users. Based on the audit result, the TPA could release an audit report to user.

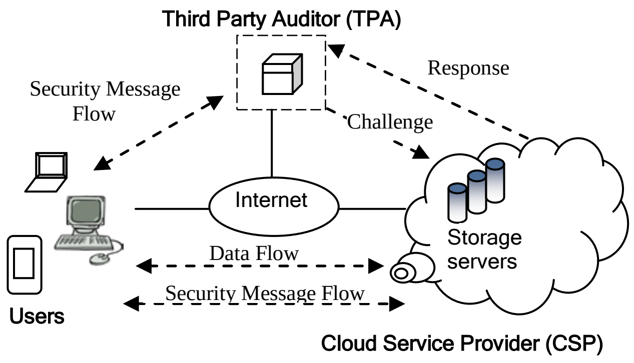


Fig.1. Cloud Data Storage Model

Throughout this paper, terms verifier or TPA and cloud server or CSP are used interchangeably

In cloud data storage model, the user stores his data in cloud through cloud service provider and if he wants to access the data back, sends a request to the CSP and receives the original data. If data is in encrypted form that can be decrypted using his secrete key. However, the data is stored in cloud is vulnerable to malicious attacks; it would bring irretrievable losses to the users, since their data is stored at an untrusted storage servers. It doesn't matter that whether data is encrypted or not before storing in cloud and no matter what trust relations the client and the server may have a priori share. The existing security mechanisms need to reevaluate. Thus, it is always desirable to need an efficient and secure method for users to verify that whether data is intact? If user does not have the time, he assigns this task to third party auditor. The auditor verifies the integrity of data on behalf of users.

3. 2. Security Threats

In this paper, we are considering two types of attacks for cloud data storage those are: Internal Attacks and External Attacks.

3.2.1. Internal Attacks: These are initiated by malicious Cloud Service Provider (CSP) or malicious users. Those are intentionally corrupting the user's data inside the cloud by modifying or deleting. They are also able obtain all the information and may leaked it to outsiders.

3.2.2. External Attacks*:* these are initiated by unauthorized parties from outside the cloud. The external attacker, who is capable of comprising cloud servers and can access the user's data as long as they are internally consistent i.e. he may delete or modify the customer's data and may leaked the user private information.

3.3. Design Goals

We have designed an efficient and secure storage protocol to ensure the following goals. These goals are classified into two categories: Efficiency and Security Goals.

3.3.1. Efficiency

The following efficiency requirements ought to be satisfied for a proposed scheme of practical use of cloud storage:

Low computaion overhead: It includes the initialization and verification overheads of the verifier and the proof generating overheads of the server. It means that the proposed scheme should be efficient in terms of computation.

Less communication overhead: It refers to the total data transferred between the verifier and server. It means that the amount of communication should be low.

Low storage cost: It refers to the additional storage of client and server required by the scheme. It means that the additional storage should be low as possible.

3.3.2. Security

In this paper, we are considering two security requirements, which are needs to be satisfied for the security of proposed scheme:

Confidentiality: Confidentiality refers to only authorized parties or systems having the ability to access protected data.

Integrity: Data Integrity refers to the protection of data from unauthorized deletion, modification or fabrication**.** Further, detects any modifications to data stored in cloud.

3.4. Notations and Permutations

- F - the data file to be stored in cloud, the file F is divide into *n* blocks of equal length: $m_1, m_2, \ldots, m_n$, where $n=[|m|/l]$.
- $f_{key}(.)$- Sobol Random Function (SRF) indexed on some *key*, which is defined as $f : \{0,1\}^* \times key \text{-} \{0,1\}^{\log_2 n}$.
- $\pi_{key}(.)$–Sobol Random Permutation (SRP) which is defined as $\pi : \{0,1\}^{\log 2(l)} \times key – \{0,1\}^{\log 2(l).}$

Elliptic Curve Cryptography over ring Z_n:

Let *n* be an integer and let *a*, *b* be two integers in Z_n such that $gcd(4a^3+27b^2, n)=1$. An elliptic curve $E_n(a, b)$ over the ring Z_n is the set of points$(x, y) \in Z_n \times Z_n$ satisfying the equation: y^2+ax+b, together with the point at infinity denoted as O_n.

4. Efficient and Secure Storage Protocol

To ensure the confidentiality and integrity of data stored in cloud, we propose an Efficient and Secure protocol. Our scheme is designed under the Elliptic Curve Cryptography [30, 38] construction and use of Sobol sequence to verify the integrity of storage data randomly. This protocol consists of three phases, namely Setup, Verification and Dynamic Data Operations and Verification. The three process model is depicted in fig.2. The construction of these phases is presented briefly as follows:

4.1. Setup

In this phase, the user pre-processes the file before storing in cloud. The Setup phase consists of three algorithms, those are: 1) KeyGen
2) Encryption 3) MetadataGen.

4.1.1. KeyGen

In this algorithm, the user generates private key and public key pair using ***algorithm 1***, it takes *k* as input and generates private key and public key pair as output as follows: the given security parameter *k* (k>512), user chooses two large primes p and q of size k such that p≡q≡ 2 (mod 3). Then compute

$$n=pq \quad (1)$$

and

$$N_n=lcm(p+1,q+1). \quad (2)$$

where N_n is a order of elliptic curve over the ring Z_n denoted by $E_n\ (0, b)$, and *b* is a randomly chosen integer such that $gcd(b, n)=1$ and compute *P* is a generator of $E_n(0, b)$. It outputs public key $PK= \{b, n, p\}$ and private key $PR=\{ N_n)\}$.

Algorithm 1: KeyGen

```
1.  Procedure: KeyGen(k) ←{ PK,PR}
2.  Take security parameter k (k>512)
3.  Choose two random primes p an q of size k:
    p≡q≡ 2 (mod 3)
4.  Compute n=pq
5.  Compute $N_n$ = lcm(p+1, q+1)
6.  Generate random integer b<n, gcd(b, n)=1
7.  Compute P, is a generator of $E_n(0,b)$
8.  Private key PR={ $N_n$ }
9.  Public key PK={n, b, P}
10. end procedure
```

4.1.2. Encryption

To ensure the confidentiality of data, the user encrypts the each data block m_i in the file *F* using ***algorithm 2***, it takes m_i, keyed Sobol Ranodom Function(SRF) and secrete random parameter *s* as inputs and produce m'_i as output as follows:

$$F=\{m_1, m_2,...m_n\}=\{m_i\}_{1\le i\le n} \quad (3)$$

$$F'=m'_i=m_i+f_k(s) \quad (4)$$

where *s* is random of size *l*.

Algorithm 2: Encryption

```
1. Procedure : Encryption($m_i$ , s)←m'$_i$
2. for 1 to n
3.     Compute $m_i'=m_i+f_k(s)$
4. end for
5. end procedure
```

4.1.3. MetadataGen:

After encrypting the data, the user computes a metadata over encrypted data to verify the integrity of data using ***algorithm 3***, which takes m'_i, public key and private key as inputs and produce metadata T_i as output:

$$T_i \leftarrow m'_i\ P(mod\ N_n)) \quad (5)$$

where $P\varepsilon\ E_n(0, b)$

Algorithm 3:MetadataGen

```
1. Procedure: MetadataGen(m'$_i$ ,n, b, P) ←T$_i$
2. for 1 to n
3.  Compute $T_i \leftarrow m'_i$ P(mod $N_n$))
4. end for
5. end procedure
```

After computation of metadata, the user sends metadata, public key to the TPA for later verification and sends file *F'* to cloud servers for storage.

4.2. Verification Phase

Once data has stored in cloud, in order to ensure the integrity of data, our scheme entirely relies on verification phase. To verify the integrity of data, the verifier first creates a challenge and sends to the server. Upon receiving a challenge from the verifier, the server computes a response as integrity proof and return to the verifier. It consists of three algorithms: 1) Challenge,
2) ProofGen 3) CheckProof .

4.2. 1. Challenge

The verifier creates a challenge by running ***algorithm 4,*** it takes k_{SRF},j, and Q as input and return *chal* as output as follows: the verifier chooses a random keys k_{SRF} and k_{SRP} using Sobol sequence and computes random indices $1\le i_j\le n$ (j= $1_{,....,}$c) of the set[1,n], where

$$c=\pi_{k_{SRP}}(c) \quad (6)$$

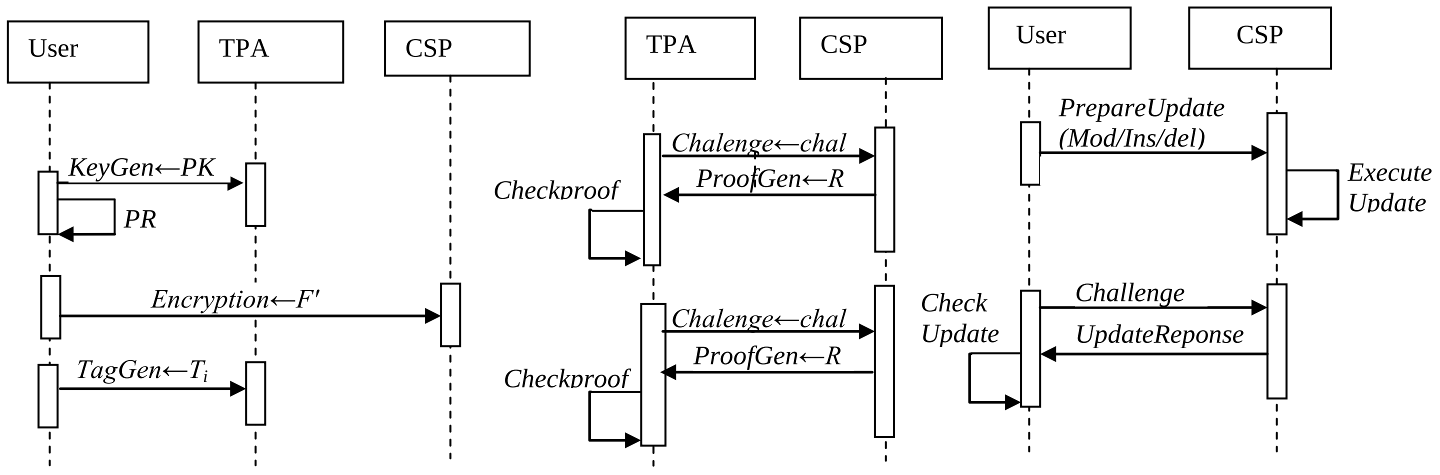
which prevents the server from anticipating which blocks will be queried in each challenge. The verifier also generates a fresh random value *r* to guarantee that the server does not reuse any values from the previous challenge and computes

$$Q=rP. \quad (7)$$

Then, verifier creates the challenge *chal*={ k_{SRF}, j, Q} , and sends to the server.

Algorithm 4: Challenge

```
1. Procedure: Challenge($k_{SRF}$,j,Q) ← chal
2. Generates a random keys $k_{SRF}$, $k_{SRP}$ and fresh
     random value using Sobol Sequence.
3. Compute $c=\pi_{k_{SRP}}$ (c)
4. Compute Q=rPε $E_n$(0, b)
5. Create challenge chal={ $k_{SRF}$, j, Q}
6. end procedure
```



(a) Setup Phase **(b) Verification Phase** **(c) Dynamic Data operations and Verification Phase**

Fig. 2. Efficient and Secure Storage Processing Model

4.2.2. ProofGen

Upon receiving a challenge from the verifier, each server computes a response as integrity proof using ***algorithm 5***, it takes encrypted data m'$_i$, challenge *chal* as inputs and produce response R as output as follows: first, it generates random numbers using Sobol random Function (SRF) i.e.

$$a_j = f_{k_{SRF}}(j) \quad (8)$$

Then compute $b = \sum_{j=1}^{c} a_j m'_{i_j}$ (9)

where $1 \le i_j \le n$

Later, computes a response $R = bQ \bmod n$ (10)

$$= \sum_{j=1}^{c} a_j m'_{i_j} Q \bmod n$$

$$= \sum_{j=1}^{c} a_j m'_{i_j} rP \bmod n$$

$$= r(\sum_{j=1}^{c} a_j m'_{i_j} P \bmod n)$$

Algorithm 5: ProofGen

```
1. Procedure: ProofGen(m'_i , k_SRF, Q)←R
2. Generates a n random numbers using k_SRF
3. for 1 to n
4. Generate $a_j = f_{k_{SRF}}(j)$
5. end for
6. compute $b = \sum_{j=1}^{c} a_j m'_{i_j}$
7. compute R=bQ mod n
8. end procedure
```

4.2.3. CheckProof

After receiving a response from the server, the verifier checks the integrity using ***algorithm 6***, it takes public key *pk*, challenge query *chal***,** and proof *R* as inputs and return output as 1 if the integrity of file is verified as successfully or 0 as follows: the verifier re-generates random numbers using Sobol Random function i.e.

$$a_j = f_{k_{SRF}}(j)$$

Then compute $S = \prod_{j=1}^{c} a_j T'_{i_j} \bmod n$ (11)

$$R' = rS \bmod n \quad (12)$$

Now, verifier checks whether

R'=R, (13)

if response is valid, then it returns 1 otherwise 0.

Algorithm 6: CheckProof

```
1. Procedure: CheckProof(T'_i , r, k_SRF, n)←R'
2. Generates a n random numbers using key k_SRF
3. for 1 to n
4. Generate $a_j = f_{k_{SRF}}(j)$
5. end for
6. compute $S = \prod_{j=1}^{c} a_j T'_{i_j} \bmod n$
7. compute $R' = rS \bmod n$
8. verify if (R'=R)
9. return true
10. else
11. return false
12. end if
13. end procedure
```

4.4. Dynamic Data Operations

The proposed scheme also supports dynamic data operations at block level [33] while maintaining same security assurance, such as Block Modification (BM), Block Insertion (BI) and Block Deletion (BD). These operations are performed by the server based on the user request in the general form (BlockOP, j, m'_i), where BlockOp indicates the block operation such as BM, BI and BD. The parameter *j* indicates the particular block to be updated and m^*_i is the new block.

In order to update data in cloud, the user creates a request and sends to the server. Upon receiving an update request from the user, the server performs the particular update operation (modification/insert/delete).

Here, we show that how our scheme supports dynamic data operations efficiently:

Algorithm 7: PrepareUpdate

```
1. Procedure:PrepareUpdate←(BM/BI/BD,j, m'_i)
2. Select a update block m_j
3. if(update==modification/insert)
4. Encrypt $m'_j \leftarrow m_j + f_k(s)$
5. Compute $T_j \leftarrow m'_j P \bmod N_n$
6. Update=(BM/BI, j, m'_i)
7. else if(update==deletion)
8. Update =((BD, j)
9. Send update request to the server
10. end if
11. end procedure
```

4.4.1. Block Modification (BM):

Data modification is one of the frequently used operations in cloud data storage. Suppose, the user wants to modify the block m_j with $m'_{i,,}$ then the user runs the **algorithm 7** to do the following:

1) Create a new block m_j
2) Encrypt the new block using equation (2)

$$m'_j \leftarrow m_j + f_k(s) \quad (14)$$

3) Compute new metadata using equation

$$T_j \leftarrow m'_j P \bmod \mathrm{N}_n \quad (15)$$

4) Create update request (BM, j, m_i) and sends to the server.
5) The Metadata sends to TPA for later verification

Upon receiving an update request, the server replace the block m'_i with m'_j and construct update version of the file *F''* by running ***algorithm 8***.

Algorithm 8: ExecuteUpdate

```
1. Procedure: ExecuteUpdate←{F''}
2. if(update==modification)
3. replace m_i with m'_j in the file F'
4. update file F''
5. else if(update==insert)
6. insert m*_x before m_i or append
7. else if(update==deletion)
8. delete m_i from file F'
9. update the file F''
10. move all blocks backward after i^th block
11. end if
12. end procedure
```

4.4.2. Block Insertion (BI)

In this operation, the user wants to insert a new block m* after position j in the file F'= {m'_1,..,m'_n}. The block insertion operation changes the logical structure of the file; the proposed scheme can perform the block insertion operation without re-computing metadata of all blocks that have been shifted after inserting a block, because block index is not included in the metadata. To perform an insertion of a new block m* after position j in a file, the user runs **algorithm 7** to do the following:

1. Create a new block m^*_j
2. Encrypt the new block

$$m'_j \leftarrow m^*_j + f_k(s) \quad (16)$$

3. Compute new metadata

$$T^*_j \leftarrow m'_j P \bmod N_n \quad (17)$$

4. Create update request (BI, j, m'_i) and sends to the server.
5. The Metadata sends to TPA for later verification

Upon receiving the update request, the server replace the block m'_j with m'_j and construct update version of the file F'' by run the ***algorithm*** 8.

4.4.3. Block Deletion (BD)

The Block deletion operation is the opposite of insertion operation. When one block is deleted, all subsequent blocks are moved one step forward. Suppose, the user wants to delete a specific data block at position j from the file F', creates a delete request (BD, j), sends to the server and also sends request to the TPA to delete corresponding block metadata. Upon receiving a delete request from the user, the server deletes the block m'_j from the file and constructs update version of the file F''. Similarly, the TPA deletes corresponding metadata. Here, deletion of metadata do not depends on other block metadata. The detail of delete operation is given in **algorithm 8.**

4.4.4. Verification

To ensure the security of dynamic data operations, the user verifies the integrity of updated block immediately after updating as follows:

1) The user challenges the server immediately for the proof of update operation i.e.

$$Q = rP \tag{18}$$

2) Upon receiving a request from the user, the server computes a response for updated block and returns to the user:

$$R_j \leftarrow m'_j P \bmod n \tag{19}$$

3) After receiving an update response from the server, the user verifies whether response is matched with metadata of particular block by running ***algorithm 9***, if it returns true, server has been updated data successfully otherwise not.

Algorithm 9 : ***VerifyUpdate***

```
1.  Procedure: VerifyUpdate(pk, Q, R')→{1,0 }
2.     if(update==modification/insert)
3.         if($T_j$=$R_j$)
4.             return  1
5.               else
6.              return 0
7.             end if
8.       else if(update==deletion)
9.      verification directly starts from static case
10.   end if
11. end procedure
```

In next section, we analyze the security of our scheme.

5. Security Analysis

In this section, we present the formal security analysis of the proposed scheme. That means integrity and confidentiality of data stored in cloud.

5.1. Integrity

To ensure the integrity, we need three properties: Completeness, Soundness and Probability Detection. Here, we define these terms as follows: for completeness, soundness [30] and Probability Detection [24]

Completeness: After receiving a challenge from the verifier, if server honestly computes a correct integrity proof, the verifier always accepts the proof as valid.

Soundness: After receiving a challenge from the verifier, the server dishonestly computes the integrity proof by missing some data bits, the verifier accepts with negligible probability.

Probability Detection: After receiving a response from the server, the verifier check whether response is valid or not? If it is not valid, then the verifier detects the corruptions with high probability.

In our integrity analysis, we have depended on the Finding order of elliptic curve and Elliptic curve discrete logarithm problem denoted by ELDL problems.

1) Finding the order of elliptic curves:

The order of elliptic curve over the ring Z_n is: let n=pq is defined in [38,] as N_n =lcm(#E_p(a, b), #E_q(a, b)). N_n is the order of the curve, i.e. for any Pε E_n(a, b) and any integer k, such that

$$(k\, N_n+1)P=P. \tag{20}$$

If(a=0 and p≡q≡2 mod 3) or (b=0 and p≡q≡3 mod 4), the order of E_n(a, b) is equal to N_n. The given

N_n =lcm(#E_p(a, b), #E_q(a, b))= lcm(p+1, q+1) (21)

Solving N_n is computationally equitant to factoring the corresponding number *n*.

2) Elliptic Curve Discrete Logarithm Problem(ECDLP)

Consider the equation Q=rp where Q, Pε E_n(a, b) and r<n. it is relatively hard to determine r given Q and P.

Theorem 1. The proposed protocol is complete

Proof: Here, we are proving this theorem according to the definition of sound and commutative property of point multiplication in an elliptic curve [30].

we have $R' = R$

$$R' = rS \bmod n$$

$$S = \prod_{j=1}^{c} a_j T_{i_j} \bmod n \ \ where\, a_j = f_k(j)$$

$$= \prod_{j=1}^{c} (a_j m'_{i_j} P \ \bmod N_n) \bmod n$$

$$= \sum_{j=1}^{c} a_j m'_{i_j} P \ \bmod n$$

$$R' = rS \bmod n$$

$$= r(\prod_{j=1}^{c} (a_j m'_{i_j} \ P \ \bmod n)$$

$$= r(\sum_{j=1}^{c} a_j m'_{i_j} P \bmod n\,)$$

$$= R$$

From the equation (13), the protocol is **complete or valid.** Then the verifier is "probabilistically" assured that server still holds data safely. In reality, verifier only verifies that server holds the *j* [1, c] selective blocks where j is chosen randomly.

Theorem 2: The proposed protocol is sound

Proof: In this proof, we show that our protocol is sound against dishonest server based on previous transactions and pre-computed metadata. There are four

possibilities that the server can compute the integrity proof without storing the user's data:
1) The server guessed or use pre-computed value. However, guessing occurs with negligible probability and pre-computing the correct response is not possible because each time the verifier challenge the server with a fresh challenge.
2) Other option is to cheat user, the server replayed the previous response. In this case, the server would have to find *r* from challenge ***chal*** to compute the correct proof. since *r* is chosen randomly, finding *r* is hard based on ELDL problem.
3) Another option for the server to cheat user, he has an algorithm to compute m'_i *mod* N_n with inputs instead of storing $m'_i[1 \le i \le n]$. But this option is not possible, because, the server cannot compute N_n based on the hardness of solving the order of elliptic curve $E_n(0, b)$ as we discussed above.
4) Last option for server is, if the server does not store the data $\{m'_i\}$ and it may try to collude with the other servers for storing the same data. However, this option is not feasible, since data stored at each server is securely encrypted using Sobol Random Function (SRF). The *f* is a keyed one-way function and *s* is a secrete parameter, so, no one except the user can retrieve the original data m_i from m'_i.

All these options lead to contradiction; so the server cannot compute response without storing the data. Hence, our protocol is complete.

Probability Detection

Here, we investigate how the probabilistic nature of the proposed protocol makes it possible to enforce integrity. We are making the following assumptions:

- The verifier's random selection of indexes is uniform, i.e., for n blocks, the probability to pick any block is 1/n.
- If an attacker removes a portion d/n of data from the storage file, this portion is referred to as the corruption of the file.
- The verifier performs on average c challenges: 1≤ c ≤n (n is the number of blocks) and detects the corruption with high probability.

The detection probability P_d of disrupted blocks is an important parameter to guarantee that these blocks can be modified or detected in a time. We have detection of probability is:

$$P_d = 1 - \left(1 - \frac{d}{n}\right)^c \quad (22)$$

For a given probability of detection of a data corruption, it is possible to probabilistically determine the average number of challenges that the verifier should perform to achieve the probability of detection. The number of challenges *c* can be derived as follows[30]:

$$c = \log_{1-\frac{d}{n}}(1 - P_d) \quad (23)$$

Fig. 3 plots P_d for different values of *n, c, d.* To understand the importance of Fig. 3, suppose that if a fraction of the data file is corrupted, the Sobol' sequence achieves detection with high probability in a few number of blocks to challenge, while pseudorandom data requires more blocks and even sometimes it may not detect the corruptions, since it do not covers the whole data in the file while verifying the data integrity. For example if *d=1%n* (data corruption) is corrupted, the proposed scheme using sobol sequence detects corruption with 99% in 4%n blocks whereas existing probabilistic checking methods using pseudorandom sequence requires 10%n blocks and sometimes these blocks may not detect the corruption. Since, sobol sequence is more uniform than pseudo random sequence.

Therefore, the proposed method is more secure and efficient than existing probabilistic remote data checking methods.

Monte-Carlo Results.

Now, we turn to Monte-Carlo simulation to determine uniformity of random sequences. For the goodness of random numbers, we calculated the Monte Carlo integration using random numbers. The integration of a function f(x) in the s-dimensional unit cube I^s. we are in fact calculating the average of the function at a set of randomly sample points. Where there are N sample points in the integral is:

$$V = \frac{1}{N}\sum_{i=1}^{N} f(x') \quad (24)$$

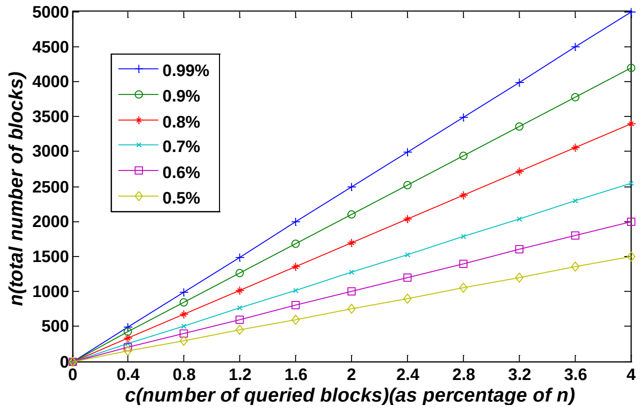
Where v is used to denote the approximation to the integral and x^1, x^1,.. ,x^N are the N, s-dimensional sample points. The Monte Carlo integration of V sampling in the region -1<x'<1 for the two cases: uncorrelated random numbers (pseudorandom sequence) and Sobol sequence. If pseudo-random sequence is used, the points x' will be independently and identically distributed, the estimate the expected error of integral is $N^{-1/2}$, while sobol sequence is used, whose fractional error decreases of N^{-1}. In Figure 4 we presented for calculation of six dimensional integral is:

$$I = \int_0^1\int_0^1\int_0^1\int_0^1\int_0^1\int_0^1\int_0^1 \prod_{i=1}^{6} (i\cos(ix_i))\, dx_1\, dx_2\, dx_3\, dx_4\, dx_5\, dx_6 \quad (25)$$

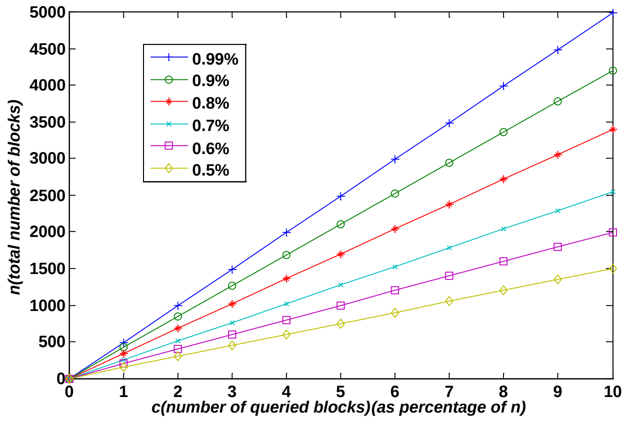
The exact value of integral is:

$$I = \prod_{i=1}^{6} \sin(i) \quad (26)$$

Fig.4 shows that the pseudorandom sequence gives worst performance, whilst Sobol Sequence gives rapid convergence to the solution. To conclude that it has been shown that Sobol sequence can evaluate integrals more efficient than pseudorandom sequences.



(a) z=1%*l* using Sobol Sequence



(b) z=1%*l* using pseudorandom sequence

Fig. 3: The detection probability p_d against data corruption. We show p_d as a function of *l*(total number of rows) and r(the number of rows queried by the user, shown as percentage of *l*) for value of z(the number of rows modified by the adversary).

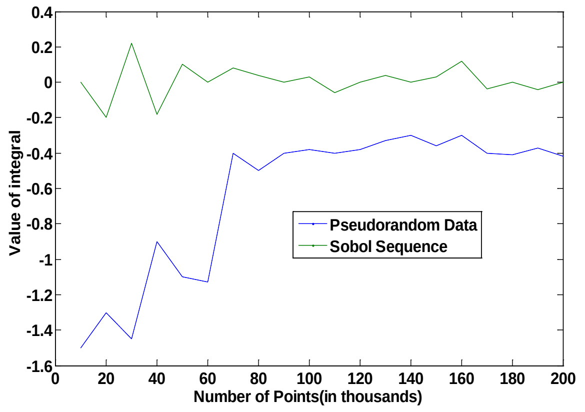


Fig. 4. Monte Carlo simulation using random numbers

5.2. Confidentiality

Now, we analyze the confidentiality of our scheme: The stored data in cloud cannot be leaked to an malicious attackers (servers and TPA). In this analysis, we depend on the hardness of the Elliptive Curve Diffie-Hellman (ECHP) and Elliptive Curve Discrete Logarithm (ECDL) problems.

Theorem 3: The proposed protocol is confidential against data leakage to attacker.

We prove this theorem under different attacks:

1) The secret parameter *s* cannot be derived by a malicious user eavesdropping on the communication link between the user and server because of Elliptive Curve Diffie-Hellman (ECDH) problem. The public parameter {b,n,P} cannot help the adversary to infer or calculate any useful information that can reveal the shared key between the user and server.

2) Suppose, If the malicious server wants to access the data from the encrypted file F'=m_i'. But it is not possible, because in order to access the encrypted data, he should need a secrete parameter, this secrete key chosen by user randomly. If server try to get the secret key by using different combinations of public parameters but fail to do so due to the ECDL problem. Hence, the server cannot learn anything from F'.

3) The TPA has $T_i \leftarrow m'_i\ P(mod\ N_n)$. If he tries to access data content from metadata, the user computes metadata over encrypted the data using secrete key. However, it is not possible because the secrete parameter chosen by the user from random. So there is no chance to TPA get secrete parameter using public key and metadata. Hence, The TPA cannot learn anything from metadata T_i .

Therefore on the basis of ECDH and ECDL problems, our protocol is confidential against data leakage.

6. Performance Analysis and Experimental Results

6.1 Performance Analysis

In this section, we analyze the performance of our scheme in terms of storage, communication and computation complexity.

Storage cost:

Here, we detail the storage cost required by the client, TPA and server.

User Side: The user needs to store the only secrete parameter. The storage cost for that is O (1).

Server Side: the server needs to be store the complete file, the cost for storage file is O (n) bits.

TPA or Verifier: the verifier needs to store metadata and public key. The metadata is a relatively smaller than original file, so storage cost for metadata is O (1).

Communication Cost:

Here, we consider the communication cost between the server and verifier during verification phase. The challenge sent by the verifier to the server, which consists of O(1) and the response(it is a small size compare to original file) sent by server to the verifier, which consists of O(1). Thus, total communication cost is O (1).

Computation Cost:

We analyze the computation cost of the user, verifier and server as follows:

User: during the setup phase, the user generates a private key and public key whose cost is O(1) . Then, to encrypt a file, the user needs to perform integer addition, its cost is O(n). Finally, computes the metadata by performing n-bit point multiplications whose cost is O(1). Hence, total computation cost of the user is: O (1).

Verifier: During the verification phase, the TPA or verifier needs to generate three random numbers ⟨k_{SRF},j, r⟩, then compute $c = \pi_{k_{SRP}}$ ***(c)*** and Q = rP, whose cost is O(1). Again, after receiving the response, the verifier re-generates $\{a_j\}$ *j=[1,c]*, the computation cost of each $a_j m'_{i_j}$ corresponds to the sum of point multiplication of two bits. Finally, the verifier computes R', the cost of R' is a two point multiplications plus sum of 2 bit integer plus generating random numbers cost, which is O(1) respectively. Hence, the total computation cost at verifier side is O(1).

Server Side: During the verification phase, the server needs to generate n-Sobolrandom b-bit integers a_i , then it computes $b = \sum_{j=1}^{c} a_j m'_{i_j}$ and R $= r\sum_{i=1}^{n} a_j m'_{i_j} P \bmod n$ The computation of each $a_j m'_{i_j}$ corresponds to the sum of point multiplication of two bits. The computation cost of $a_j m'_{i_j}$ is O(1). Next, the server computes a proof, which consists of point multiplications in ***ProofGen*** algorithm, its cost is O (1). The total computation cost of server for generating integrity proof (response) is O(1).

In table 1, we summarized the storage, communication and computation costs.

Table 1: Summary of Storage, Communication and Computation cost of Proposed Protocol

Storage Cost			Communication Cost		Computation Cost	
Verifier	Server	Verifier	Server	User	Verifier	Server
O(1)	O(n)	O(1)	O(1)	O(1)	O(1)	O(1)

6.2. Experimental Results

In this section, we present the experimental results of our protocol. All experiments conducted using C++ on system with dual core 2-GHZ processor and 4GB RAM running Windows 2007. In our implementation, we use MIRACL library version 5.4.2 to achieve better security work on elliptic curve with 160-bit group order instead of RSA on 1024 bits. Here, we are measuring total time for computation cost of the verifier and server using ECC and RSA respectively.

$$Speedup = \frac{RSA - ECC}{RSA} * 100 \qquad (27)$$

Then, we compare computation cost of our protocol with RSA-based remote data checking protocols, which includes the verifier, server and user computation costs and presented results in table 2, 3 & 4.

Table 2: Computation Cost at Verifier using RSA [13] and ECC based schemes.

File Size	Verifier side using RSA[33]	Verifier Side using ECC	Speedup
10MB	424.37 ms	316.26 ms	25%
20MB	482.81 ms	342.43 ms	29%
30MB	561.62 ms	376.03 ms	32%
40MB	641.46 ms	415.09 ms	35%
50MB	743.64 ms	465.13 ms	38%

Table 2 shows that the total computation cost of verifier for our proposed scheme is faster than existing RSA based scheme [33]

Table 3: Computation Cost at Server with RSA based scheme and ECC scheme

l(bits)	Server Side with RSA[33]	Server Side with ECC	Speedup(%)
10MB	388.01 ms	275.11 ms	29%
20MB	447.62 ms	312.43 ms	30%
30MB	508.39 ms	348.21 ms	31%
40MB	562.67 ms	381.21 ms	32%
50MB	625.16 ms	418.76 ms	33%

Table 3 shows that the total computation cost of the server for proposed scheme is faster than existing RSA based scheme [33].

Table 4: Metadata Computation Cost at user with RSA and ECC based schemes

l(bits)	Server Side with RSA[33]	Server Side with ECC	SpeedUp(%)
10MB	244.11 ms	183.06 ms	25%
20MB	296.41 ms	218.32 ms	26%
30MB	352.53 ms	253.38 ms	28%
40MB	403.17 ms	289.63 ms	29%
50MB	467.26 ms	323.92 ms	30%

Table 4 shows that the total computation cost of metadata at user side in our scheme is faster than existing RSA based scheme [33]

6.3. Comparison with Existing Schemes

We compared our scheme with existing RSA based verification schemes and probabilistic verifications schemes

Most of the schemes that use RSA based verification but the key length for secure RSA use as increased over recent years and this put a heavier processing burden on applications using RSA. To avoid this problem, we proposed an ECC based verification scheme. The principal of ECC compared to RSA is that it appear to offer equal security for a far smaller key size, thereby it reduced the computation overhead.

Table 5: Comparisons between Proposed Protocol and selective Existing Protocols

	RDC [13]	**C.wang [21]**	**Q.wang [23]**	**Yan[24]**	**C.wang [30]**	**Hao[32]**	**Syam[27]**	**Prposed protocol**
Type of Guaranty	Prob	Prob	Prob	Prob	Prob	Deter	Prob	**Prob**
Integrity	Partial	Partial	Partial	Partial	Partial	Yes	Yes	**Yes**
Confidentiality	no	no	no	no	Partial	Partial	no	**Yes**
Public Verifiability	no	Yes	Yes	Yes	Yes	Yes	no	**Yes**
Data Dynamics	no	Partial	Yes	Yes	Yes	Yes	Partial	**Yes**
Communication complexity	O(1)	O(clogn)	O(logn)	O(s)	O(logn)	O(1)	O(1)	**O(1)**
Server Computation	O(1)	O(clogn)	O(logn)	O(c+s)	O(nlogn)	O(n)	O(clogn)	**O(1)**
Verifier computation	O(1)	O(clogn)	O(logn)	O(c+s)	O(logn)	O(n)	O(clogn)	**O(1)**
Probability Detection	$O(N^{-1/2})$	$O(N^{-1/2})$	$O(N^{-1/2})$	$O(N^{-1/2})$	$O(N^{-1/2})$	$O(N^{-1/2})$	$O(N^{-1})$	$\mathbf{O(N^{-1})}$

Prob: Probabilistic **Deter:** Deterministic

Next, we compare our scheme with probabilistic verification schemes. These schemes verify the integrity of outsourced data based on pseudorandom sequence but they do not provide satisfactory integrity assurance to the users i.e. sometimes they may not detect the data corruptions in cloud. Because, pseudorandom sequence is not uniform (uncorrelated random numbers), and it will take more time to detect data corruption, so its time consuming whereas proposed protocol verifies the integrity of the data using Sobol sequence. Our scheme should detect all data corruptions with less number of blocks since sobol sequence covers the entire data in the file more uniformly than pseudorandom sequence.

Finally, the proposed protocol is private against unauthorized data leakage because, we are encrypting the data before storing in cloud. In Table 5, we summarize the comparison between the selective existing protocols and proposed protocol.

7. Conclusion

In this paper, we have studied the problem of Integrity and Confidentiality of data storage in cloud computing and proposed an efficient and secure protocol using ECC and Sobol sequence. The proposed method is mainly suitable for thin users who have less resources and limited computing capability. It satisfies the all security and performance requirements of cloud data storage. Our method also supports public verifiability that enables TPA to verify the integrity of data without retrieving original data from the server and probability detects data corruptions. Moreover, our scheme also supports dynamic data operations, which performed by the user on data stored in cloud while maintaining same security assurance. We have proved that proposed scheme is secure in terms of integrity and confidentiality through security analysis. Through, performance analysis and experimental results proved that proposed scheme is efficient. Compared with previously proposed protocols, we have also proved that proposed scheme is more secure and efficient.

Acknowledgments

The first author would like to acknowledge Prof. J.A.K. Tareen, vice-chancellor for availing good research environment and providing scholarship. The First author also would like to thank HRD ministry of India for providing the UGC-RGNF Fellowship.

References

[1] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic."Cloud Computing and Emerging IT Platforms: Vision, Hype, and Reality for Delivering Computing as the 5th Utility," Future Generation Computer Systems, vol. 25, no. 6, June 2009, pp 599–616.

[2] Amazon.com, "Amazon Web Services (AWS)," Online at http://aws.amazon.com, 2008.

[3] Apple "ICloud" Online at http://www.apple.com/icloud/what-is.html 2010.

[4] T Mather, S Kumaraswamy, and S Latif "Cloud Security and Privacy", O'REILLY Publication, first edition, sep-2009.

[5] H. Takabi, J.B.D. Joshi, and G. Ahn, "Security and Privacy Challenges in Cloud Computing Environments", Article in IEEE Security and Privacy, vol. 8, no.6, Nov-Dec. 2010, pp. 24-31.

[6] N. Gohring, "Amazon's S3 down for several hours," Online at http://www.pcworld.com/businesscenter/ articl/ 142549/amasons_down_for_sever_hours.html", 2008 .

[7] J. Kincaid, "MediaMax/TheLinkup Closes Its Doors",Online at http://www.techcrunch.com/2008/07/10/mediamaxthelinkup-closes-its-doors/, July 2008.

[8] Y. Deswarte, J.-J. Quisquater, and A. Saidane. "Remote integrity checking". In Proc. of Conference on Integrity and Internal Control in Information Systems (IICIS'03), November 2003. lausanne, Switzerland.

[9] D. L. G. Filho and P. S. L. M. Barreto, "Demonstrating Data Possession and Uncheatable Data Transfer," cryptology ePrint Archive, Report 2006/150, 2006, http://eprint.iacr.org/.

[10] G. Caronni and M. Waldvogel, "Establishing Trust in Distributed Storage Providers", In Third IEEE P2P Conference, Linkoping 03, 2003.

[11] P. Golle, S. Jarecki and I. Mironov, "Cryptographic Primitives Enforcing Communication and Storage Complexity", In proc. of Financial Crypto 2002. Southampton, Bermuda.

[12] F. Sebe. J. Domingo-Ferrer, and A. Martinez-Balleste, Y. Deswarte, and J.-J. Quisquater, "Efficient Remote Data Possession Checking in Critical Information Infrastucures", IEEE Trans. Knowledge and Data Engineering, vol. 20, no. 8, aug-2008, pp. 1034-1038

[13] G. Ateniese, R. Burns, R. Curtmola, J. Herring, L. Kissner, Z. Peterson,and D. Song, "Remote Data Checking using Provable Data Possession," ACM Trans. ACM Transactions on Information and System Security, Vol. 14, No. 1, Article 12, may 2011, pp. 12.1–12.34.

[14] A. Juels and J. Burton S. Kaliski, "PORs: Proofs of Retrievability for Large Files," *Proc. of CCS '07*, pp. 584–597, 2007.Alexandria, Va, USA.

[15] G Ateniese, S. Kamara, J. Katz, "Proofs of Storage from homomorphic identification protocols". In Proc. of ASIACRYPT '09, 2009, pp. 319-333.Tokyo, Japan.

[16] H.Shacham and B.Waters, "Compact Proofs of Retrievability", Proc.14th Int'l Conference Theory and Application of Cryptology and Information Security: Advances in Cryptology (ASIACRYPT), LNCS 5350,2008, pp.90-107. Melborne, Austrilia.

[17] Y. Dodis, S. Vadhan, D. Wichs. "Proofs of retrievability via hardness amplification". In: Proc. of TCC '09, 2009, pp.109--127.CA, USA.

[18] K. D. Bowers, A. Juels, and A. Oprea, "Proofs of Retrievability: Theory and Implementation," Cryptology ePrint Archive, Report 2008/175, 2008 http://eprint.iacr.org/.

[19] K. D. Bowers, A. Juels, and A. Oprea, "HAIL: A High-Availability and Integrity Layer for Cloud Storage," Cryptology ePrint Archive, Report 2008/489, 2008, http://eprint.iacr.org/.

[20] T. S. J. Schwarz and E. L. Miller, "Store, Forget, and Check: Using Algebraic Signatures to Check Remotely Administered Storage,"*Proc.of ICDCS'06*,2006,pp.12–21, Lisboa, Portugal.

[21] R. Curtmola, O. Khan, and R. Burns. "Robust remote data checking". In: Proc. of StorageSS '08, 2008, pp.63-68, Virginal, USA.

[22] G. Ateniese, R. D. Pietro, L. V. Mancini, and G. Tsudik, "Scalable and Efficient Provable Data Possession," *Proc. of SecureComm '08*, pp. 1–10, 2008.Istanbul, Turkey.

[23] C. Wang, Q. Wang, K. Ren, N. cao and W. Lou , "Towards Secure and Dependable Storage Services in Cloud Computing", Accepted for publication in future issue of IEEE Trans. Service Computing. DOI:10.1109/TSC.2011.24.

[24] C. Erway, A. K¨upc¨u, C. Papamanthou, and R. Tamassia, "Dynamic provable data possession," in CCS'09, ACM, 2009, pp. 213–222, Chicago, USA.

[25] Q. Wang, C. Wang, K. Ren W. Lou, and J. Li, "Enabling public verifiability and data dynamics for storage security in cloud computing," IEEE Trans. Parallel and Distributed Computing.VOL.22, NO.5, May 2011, pp.847-859

[26] Yan Zhu, Huaixi Wang, Zexing Hu, Gail-J. Ahn, Hongxin Hu, Stephen S. Yau, "Dynamic Audit Services for Integrity Verification of Outsourced Storages in Clouds," Proc. of the 26th ACM Symposium on Applied Computing (SAC), Tunghai University, TaiChung, Taiwan, March 21-24, 2011.

[27] P. Syam Kumar, R. Subramanian, "Homomorpic Distributed Verification Ptorotocol for Ensuring Data Storage in Cloud Computing". International Journal of Information, VOL. 14, NO.10, OCT-2011, pp.3465-3476.

[28] M. A. Shah, M. Baker, J. C. Mogul, and R. Swaminathan, "Auditing to Keep Online Storage Services Honest," *Proc. 11th USENIX Workshop on Hot Topics in Operating Systems (HOTOS '07)*, 2007, pp. 1–6, CA, USA.

[29] M. A. Shah, R. Swaminathan, and M. Baker, "Privacy-preserving audit and extraction of digital contents," Cryptology ePrint Archive, Report 2008/186, 2008, http://eprint.iacr.org/.

[30] N. Oualha, M. Onen, Y. Roudier,., "A Security Protocol for Self-Organizing Data Storage". Tech. Rep. EURECOM+2399, Institut Eurecom, 200 8, France.

[31] C. Wang, Q. Wang, K. Ren, and W. Lou, "Privacy-preserving public auditing for data storage security in cloud computing," In Proc. of IEEE INFOCOM'10, , March 2010. San Diego, CA, USA.

[32] Z. Hao and N. Yu, "A multiple-replica remote data possession checking protocol with public verifiability," in Second International Symposium on Data, Privacy, and E-Commerce , 2010.Buffalo, Niagara Falls.

[33] Z. Hao, S. Zhong, and N. Yu, "A Privacy-Preserving Remote Data Integrity Checking Protocol with Data Dynamics and Public Verifiability", Accepted for publication in future issue of IEEE Trans. Knowledge and Data Engineering, DOI: 10.1109/TKDE.2011.62

[34] A. F. Barsoum and M. A. Hasan, "Provable possession and replication of data over cloud servers," Centre For Applied Cryptographic Research (CACR), University of Waterloo,Report2010/32,2010,http://www.cacr.math.uwaterloo.ca/ techreports/2010/cacr2010- 32.pdf.

[35] L. Chen, G. Guo, "An Efficient Remote Data Possession Checking in Cloud Storage", International Journal of Digital Content Technology and its Applications. Volume 5, Number 4, April 2011, pp.43-50.

[36] J. Yang, H. Wang, J. Wang1, C. Tan and D. Yu, "Provable Data Possession of Resource-constrained Mobile Devices in Cloud Computing" JOURNAL OF NETWORKS, VOL. 6, NO.7, JULY 2011, pp.1033-40.

[37] V. Miller, "Uses of elliptic curves in cryptography", advances in Cryptology, Proceedings of Crypto'85, Lecture Notes in Computer Science, 218 Springer-Verlag, pp.417-426. 1986.

[38] K. Koyama, U. Maurer, T. Okamoto, and S. Vanstone, "New Public-Key Schemes Based on Elliptic Curves over the Ring Zn", Advances in Cryptology - CRYPTO '91, Lecture Notes in Computer Science, Springer-Verlag, vol. 576, Aug 1991, pp. 252-266,.

[39] Brately P and Fox B L "Algorithm 659: Implementing Sobol's Quasi-random Sequence Generator" ACM Trans. Math. Software 14 (1), (1988) , pp. 88–100.

1P Syam Kumar: is currently Ph.D student (Computer Science), in department of Computer Science, School of Engineering and technology, Pondicherry University, Puducherry, India. He received M.Tech Degree in 2006 from the department of Computer Science and Technology in Andhra University, India, and received B.Tech (graduation) in 2003 from CSE department, Vagdevi college of Engineering, JNTU Warangal, India. He is especially interested on Network Security, Cloud Computing, Distributed Systems etc.

2R. Subramanian: is Professor of Computer Science department, School of Engineering and technology, Pondicherry University, Puducherry, India. He received his Ph.D degree of The Department of Mathematics, **IIT** Delhi, India in 1989. He received a M.Sc . degree in 1984 from IIT Delhi, India and he received B.Sc. form Madurai Kamaraj University in 1982, Tamilanadu, India. He is especially interested in Parallel & Distributed Algorithms, Cloud Computing, Evolutionary Algorithms, Robotics, etc.