International Journal for Research in Engineering Application & Management (IJREAM)
ISSN : 2454-9150 Vol-04, Issue-06, Sep 2018

Concurrent Trie Hashing Based Checksum Signature Integrity Verification For Secured Cloud Data Storage

[1]P. Jayasree , [2]Dr. V. Saravanan

[1]Assistant Professor, [2]Associate Professor, Hindusthan College of Arts and Science, Coimbatore, Tamilnadu, India,

[1]*kandasamysree@gmail.com*, [2]*vsreesaran@gmail.com*

ABSTRACT - Cloud computing is the computing model which affords user requested services through the internet. Several users store their data on a cloud server for enhancing their data security. The data stored in the cloud needs to be preserving integrity. However, the existing technique does not improve the data integrity rate and minimize the complexity. In order to improve the data integrity rate with minimal space complexity, Concurrent Trie Hashing based Checksum Signature Integrity Verification (CTH-CSIV) Method is developed. The CTH-CSIV method comprises two steps namely Concurrent Trie Hashing and Checksum Signature Integrity Verification. Concurrent Trie Hash stores the cloud user data to the server using a hash function with minimal space complexity. After that, Checksum Signature for each data is created which is kept secretly by cloud user. After storing the cloud data in the server, the Checksum Signature Integrity Verification is carried out for checking the data integrity. The checksum is a small-sized datum obtained from a block of input data for detecting errors in the cloud storage. If the generated checksum present data matches with the previously generated checksum, then the data does not alter or corrupted. This helps to obtain high security on cloud data storage. Experimental evaluation of CTH-CSIV method is carried out on factors such as space complexity, processing time and data integrity rate with respect to a number of cloud user data. The experimental results reveal that the CTH-CSIV method attains 18 % data integrity rate and also reduces space complexity of secured cloud data storage by 20 % than the other state –the-art-methods.

Keyword: Cloud computing, cloud data storage, data integrity, Concurrent Trie Hash, Checksum Signature Integrity Verification.

I. INTRODUCTION

Cloud Computing offers the various applications in terms of services through the web. During the service provisioning, security plays a major role for protecting the cloud user data. In cloud computing, the significant concern to be addressed to guarantee integrity. The integrity offers the assurance that the data is high quality and unmodified over the entire process. After storing the data on the server, the cloud user trusts their data in a secured manner. But sometimes the user's data may be altered or removed. So, the major concern is the data integrity checking at untrusted servers. In order to address the integrity problem, several methods have been introduced in this section.

Identity-based Remote Data Integrity Checking (ID-based-RDIC) scheme was presented in [1] for secure cloud data storage. The data integrity was not achieved effectively with less complexity. An identity-based proxy-oriented data uploading and remote data integrity checking (IDPUIC) scheme [2] was introduced in public cloud. Depending on client authorization, IDPUIC scheme recognizes private remote data integrity checking, delegated remote data integrity checking and public remote data integrity examination. But, it failed to use a hash function for integrity checking. A Cryptographic Mechanism was developed in [3] for data security and privacy preservation in cloud storage. The mechanism does not consider the processing complexity as an essential requirement for security in cloud storage.

An attribute-based cloud storage system was introduced in [4] for preserving the data privacy from storage servers. It takes high computational time with less data privacy level. A Trusted Third Party with Symmetric Encryption (TTPSE) scheme was designed in [5] for secure cloud storage. The scheme failed to use some kind of hashing mechanism through which searching of encrypted data in the cloud becomes difficult.

Dynamic Merkle hash B+ tree (DMBHT) was developed in [6] for data storage with minimum time consumption and for verification at server and client side. But it failed to improve reliability and scalability during the integrity verification. A fine-grained and heterogeneous proxy re-encryption (FH-PRE) system was introduced in [7] to improve the confidentiality of cloud data. By using the FH-PRE system in the cloud, cloud data was stored in cloud server with high security and shared in a fine-grained

 DOI : 10.18231/2454-9150.2018.0735

manner. Though the system improves the confidentiality, the integrity of the cloud data remained unaddressed.

To evade cloud storage servers from the data modification, a threshold encryption method combined with a protected decentralized erasure code was developed in [8]. The method obtains high data robustness, confidentiality and integrity. The complexity involved during the data storage was not minimized. An identity-based data storage and integrity verification protocol was introduced in [9] untrusted cloud environment. The security and performance analysis of this protocol does not improve.

A lattice and Bloom filter methods were introduced in [10] for reducing the consumption of cloud storage space and dynamic integrity. The method does not improve the cloud storage platform since it consumes more memory consumption.

The certain issues identified from the above said reviews such as high time and space complexity, lack of integrity and security, reliability, failed to use a hash function for improving the data security and so on. In order to address the above-said issues, an efficient method called Concurrent Trie Hashing based Checksum Signature Integrity Verification (CTH-CSIV) is developed.

The contributions of this paper are summarized as follows, Concurrent Trie Hashing based Checksum Signature Integrity Verification (CTH-CSIV) method is introduced to improve the data integrity rate and minimize space complexity as well as processing time. For achieving these contributions, two processes are carried out such as secured cloud storage and data integrity verification. At first, the cloud user stores the data on cloud server using hash value for improving the storage security. The checksum signature is generated for each data. Concurrent Trie Hashing is a tree structure where each node is labeled with the key-value pair for identifying the node location in the tree. The two dynamic operations insertion and removal is exploited for adding and removing the data from the tree. This helps to minimize complexity and processing time.

Secondly, the cloud user generates the checksum signature for the stored data. Then the cloud user checking the data integrity by matching the newly generated checksum and previously generated checksum. If two checksums are matched, then the data is not modified. This helps to improve data integrity rate.

The paper is ordered as follows. In Section 2, we review the related works. In Section 3, the proposed CTH-CSIV Method is described with neat diagram. Experimental evaluation of proposed CTH-CSIV Method and state-of-art methods are described with the dataset in section 4. Report on the experimental performance and implementation results in section 5. Section 6 concludes the paper.

II. RELATED WORKS

An integrality verification of completeness and zero-knowledge property (IVCZKP) method was introduced in [11] for integrity verification. But, Hash-based secured storage was not performed to minimize the space complexity. An encrypted Bloom filter was developed in [12] for data integrity verification by identifying the data corruption. The strategy does not guarantee the data completeness and freshness. A privacy-preserving remote data integrity-checking protocol was introduced in [13] for enhancing security of cloud data storage. The protocol does not lessen the processing time.

Data provenance method was developed in [14] for validating the data integrity by a cloud user. But, the hashing process was not carried out for improving the security. A self-adaption fault-tolerant mechanism based on access frequency (SFMAF) was introduced in [15] for improving the secured cloud storage. The performance of the integrity was not carried out to improve the storage capability. An effective and protected dynamic auditing protocol was presented [16] for data storage in cloud computing. But, the data integrity rate was not improved.

A storage enforcing remote verification approach was designed in [17] that utilize polynomial hash for cloud storage verification. The scheme does not obtain high storage security. Data partitioning technique was introduced in [18] to increase cloud data storage security. But, the technique failed to provide the high security for large data. An effective and secure public verification of data integrity approach was introduced in [19]. The approach does not optimize the verification overhead. A public auditing protocol was designed in [20] for providing the security on cloud storage and verifying the data integrity. But it has high computational complexity in integrity verification.

The above-said issues are overcome by introducing a novel integrity verification technique.

III. CONCURRENT TRIE HASHING BASED CHECKSUM SIGNATURE INTEGRITY VERIFICATION

A cloud storage system provides the reliable storage service to the number of cloud users with minimum time. A cloud user stores their data in a cloud storage server, and accesses them. Because, the user no longer maintaining his data in the local repository with high security. Hence, a security for cloud storage is major issues in cloud computing. The security of the stored files is enhanced by data integrity verification. Data Integrity is the accuracy of the stored data without any alteration. It guarantees the data can only access or altered by the authorized user. In this paper, integrity verification method is developed to improve user data robustness against the malicious user. The concurrent trie hashing based checksum signature integrity verification (CTH-CSIV) method is introduced to improve the data

 DOI : 10.18231/2454-9150.2018.0735

integrity. A concurrent hash-trie is an implementation of a hash array mapped trie to add and remove the data from the cloud. Then the data integrity is verified through checksum signature verification. The architecture diagram of CTH-CSIV Method is illustrated in figure 1.

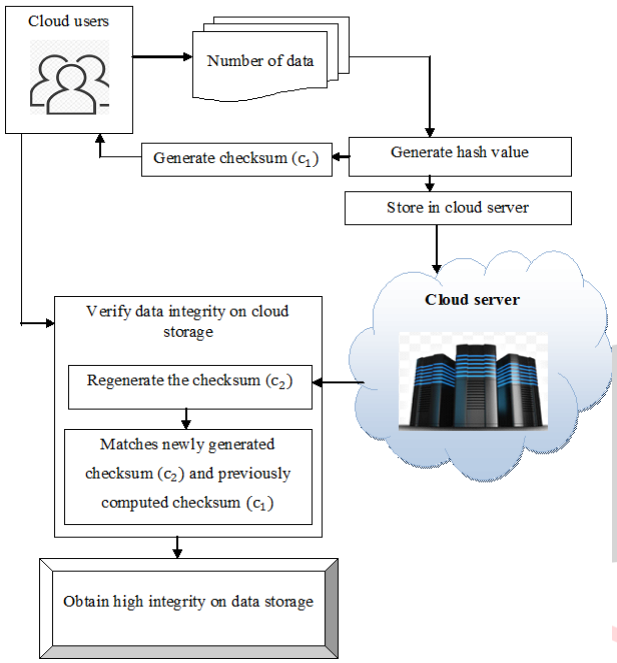

Figure 1 Architecture diagram of CTH-CSIV Method

As shown in figure 1, architecture diagram of CTH-CSIV Method is described to obtain high integrity on stored data in a cloud server. Let us consider the number of cloud users $Cu_1, Cu_2, Cu_3 \ldots, Cu_n$ sends their data $D_1, D_2, D_3, \ldots . D_n$ to store on a cloud server. The hash value is generated for each data and generates the checksum for that hash value. Then the data are stored in a cloud server. Whenever a cloud user verifies the integrity of their stored data in the cloud server, the checksum is recomputed for that hash value. Then the cloud user verifies the integrity by matching the newly computed checksum and already stored checksum at the time of data storage. If two checksum values are matched, then the cloud data does not altered by any malicious users. Otherwise, the integrity is not achieved. By this way, integrity is enhanced. The integrity verification process is explained in following sections.

3.1 Concurrent Trie Hashing based cloud data storage

The first step in the CTH-CSIV Method is to store the user data on the cloud server. Cloud storage provides the services to users for storing their data to remote servers. The Concurrent Trie Hashing technique is employed for storing the cloud user data on the server with the help of hash function. Concurrent Trie Hashing is the implementation of a hash array mapped trie to perform the concurrent insert and remove operations for cloud user data in the tree. This hashing technique is memory efficient than the other hash-based storage system. A trie is a digital tree exploited to upload a dynamic set of data. A Concurrent Trie includes number of advantages over binary search trees. The advantage of Trie is memory efficient thus minimizes the time and space complexity. The hash array mapped trie utilizes the total 32-bit space for hash values. The Concurrent Trie Hashing technique is an ordered tree structure, which is used mostly for storing the data in a compact way. A tree data structure includes a collection of nodes such as root node, branch node and leaf nodes where each node consisting a value. The root node is the top node in a tree. A branch node is a node which contains at least one children and the branch is also identified as an external node. A leaf node has no children.

To preserve the storage space of cloud server, every node in a tree contains 32 bits bitmap where each bit illustrates the occurrence of a branch node followed by an array of length equal to the Hamming weight of the bitmap. The bitmap is a bit array data structure that efficiently stores bits. Hamming weight is a number of symbols that are dissimilar from the zero-symbol of the alphabet exploited. The structure of concurrent trie is shown in figure 2.

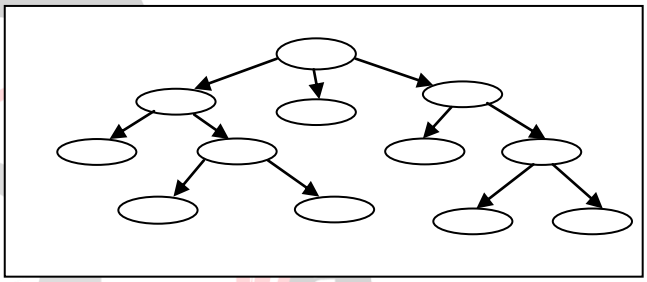

Figure 2 structure of Concurrent Trie

As shown in figure 2, Concurrent Trie based data storage is described and it has data and array blocks. Each internal node is an element array. The nodes in the tree stores key-value pairs. A key-value pair used in the Concurrent Trie includes a set of linked data items such as a key and the value. A key is a unique identifier for a number of data and the value is a pointer to identify the location of that data in a bit array. The main advantage of key-value pair is to identify the user where the data is stored in the cloud server resulting minimizing the time complexity in the data verification process. The above figure shows the root node of the tree has an empty string.

A Concurrent Trie data structure consist $'n'$ number of nodes and it is mathematically expressed as,

$$T = \{n_i | n_i \in H(D_i), \quad 1 \leq i \leq n \qquad (1)$$

From (1), where T represents the Hash Trie, n_i denotes the number of nodes in tree and $H(D_i)$ represents a hash function of data D_i. Subsequently, every node has two different sub-trees, generally represented as left and right.

$$n_i = (n_i^L, n_i^R) \qquad (2)$$

From (2), n_i^L and n_i^R indicates left and right nodes respectively. Every node uses an array to store the data with

 DOI : 10.18231/2454-9150.2018.0735

hash function. Concurrent Trie Hashing technique stores the various user data $D = D_1, D_2, D_3, \ldots . D_n$ in bits of the array using hash functions. Then, the hash value of the user data is denoted as $H = h_1, h_2, h_3, \ldots . h_n$.

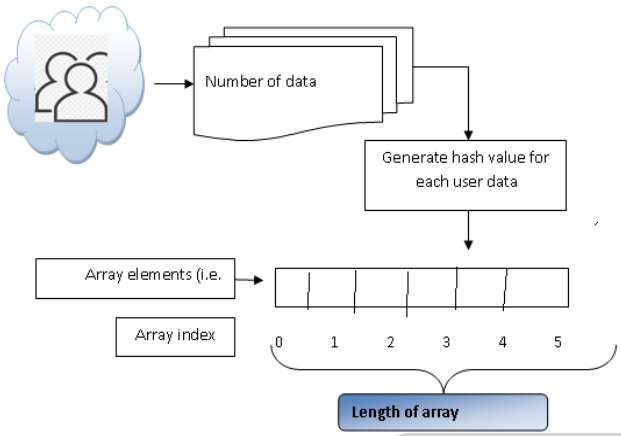

Figure 3 cloud user data storage

Figure 3 shows the Concurrent Trie used for generating the hash value for each data and stores the data in array blocks. Before the data storage, the cloud user creates checksum for each hash value. Checksums are small-sized data attained from a block of digital data for checking the entire data integrity. The checksum generation process creating checksum and store for future reference. It is done with two processes. First, the input data is reduced to a small string termed a checksum. Secondly, the checksum is stored in a database for integrity verification. The generated checksum is only known to the cloud user. The process of checksum generation is shown in figure 4.

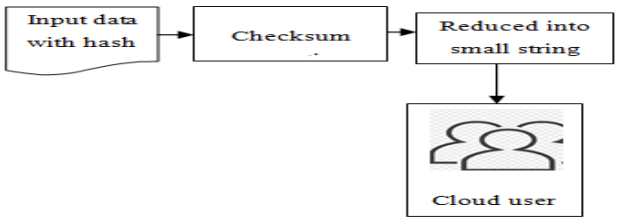

Figure 4 Flow process of checksum generation

As shown in figure 4, the flow process of checksum generation is defined to create the reduced string from the original data (i.e. data with a hash value). These generated strings are known by the cloud user but it is not stored in the cloud server.

$$C_1 \rightarrow H(D_i) \ (3)$$

From (3),C_1 denotes a created checksum for the hash value of data $H(D_i)$. The data with hash value is stored in a cloud server using Concurrent Trie. The tree is used to increases the data storage process with minimum memory usage hence it reduces the space complexity. Concurrent Trie array is a data structure includes a number of elements (i.e. cloud user data). Each data is stored by one array index.

The concurrent trie performs two operations namely insert and remove at the same time. Whenever, a cloud user wants to insert the data into a tree, the server receives user data and it performs the update operation. The insertion uses the key to find an empty entry. If an empty entry is determined, new leaf nodes are generated in the tree. Then the data is stored in leaf node which resulting extends the hash trie with the new level. Therefore, the insertion operation extends the concurrent trie with additional level.

The remove operation in concurrent trie is another major operation which ensures that the unwanted memory is freed and that the trie is kept compact. Whenever the cloud user wants to remove the data from the tree, the removal operation is performed. The remove operation eradicates need for the additional level of the trie. During the removal operation, the data which is stored in the leaf node is removed from the tree. Then the leaf node is connecting to the particular root node. As a result, the concurrent trie hashing performs both concurrent insertion operations as well as removes operations to improve the storage efficiency in a cloud server.

3.2 Checksum signature integrity verification

The cloud user wants to verify their data which hasn't been modified by untrusted parties. The cloud user data corrupted in various manners such as faulty storage media, errors in transmission, and write errors during copying or moving, software bugs, and so on. Hence, the verification is an essential process to improve the security of cloud data storage. The integrity is verified by the cloud user using checksum signature. Before the verification process, the cloud user again generated the checksum signature for stored data.

$$C_2 \rightarrow H(D_i) \ (4)$$

From (4), C_2 denotes a newly generated checksum for the stored data $H(D_i)$. if the currently generated checksum (C_2) for current data matches with the already generated checksum C_1, there is high probability that data is not accidentally altered or corrupted. If the two checksums does not match, then the process returns a verification error.

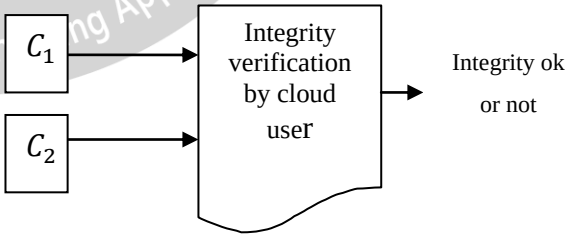

Figure 5 flow process of integrity verification

Figure 5 illustrates a flow process of integrity verification through the checksum value. The cloud user verifies the data integrity with the two checksum value. It is expressed as follows,

$$Cu = \begin{cases} integrity\ is\ achieved, & if\ C_1 = C_2 \\ integrity\ is\ not\ achieved & ,if\ C_1 \neq C_2 \end{cases} \ (5)$$

From (5), Cu denotes a cloud user. Based on the above results, the data integrity is verified to improve the security on cloud data storage. The algorithmic description of

 DOI : 10.18231/2454-9150.2018.0735

Concurrent Trie Hashing based Checksum Signature Integrity Verification is explained as follows,

Input: No. of Cloud user Data $D_i = D_1, D_2, D_3, \ldots . D_n$
Output: Improve data integrity and Reduce space complexity
1: Begin
2: For each cloud user data D_i
3: Generate hash function $H(D_i)$
4: Generate checksum $\boldsymbol{C_1}$
5: Store $H(D_i)$ to bit array in cloud server
\\ **Data Insertion operation**
6. If D_i insert to tree **then**
7. create new leaf node
8. $Insert\ (D_i, node\ (k, v))$
9. end if
\\ **Data removal Operation**
10. If D_i remove from tree **then**
11. Removes unwanted leaf node from the tree
12. $Remove\ (D_i, node\ (k, v))$
13. Connect the subtree to particular root node
14. **end if**
\\ **Checksum integrity verification**
15. **For** each $H(D_i)$
16. Cloud user generates checksum C_2
17. $\boldsymbol{If\ C_1 = C_2}$ **then**
18. D_i **not altered or modified**
19. **else**
20. D_i **is corrupted**
21. **end if**
22. end for
23:End

Algorithm 1 Concurrent Trie Hashing based Checksum Signature Integrity Verification

Algorithm 1 clearly describes the concurrent trie hashing based checksum signature integrity verification for improving the integrity of cloud storage and minimizing space complexity. The cloud user sends the request to store their data to the cloud server. The hash value for each user data is generated. Then the checksum signature is generated for each hashed data. This checksum is only known to the cloud user for verifying the data integrity. Then the hashed data is stored in the tree using bit array. Followed by, the insertion and removal operation is executed to add new data in a tree and remove the data from the tree. During the insertion operation, the cloud server creates a new leaf node with the key-value pair. Then the new data is inserted into the created leaf node. In removal operation, the particular leaf node and their key-value pair are removed. Then the subtree is connected to the particular root node. After storing the data into a tree, the integrity verification is carried out using checksum value. Whenever the cloud user verifies the data integrity, the user again regenerated the checksum for hashed data. Then the newly generated checksum is matched with a previously computed checksum. If the two checksum values are correctly matched, then the stored data is not altered. If the checksums do not match, then the stored data is corrupted. As a result, CTH-CSIV method enhances the data storage with minimum space complexity and also improves the integrity.

IV. EXPERIMENTAL SETTINGS

Experimental evaluation of CTH-CSIV Method and existing ID-based RDIC scheme [1] and IDentity-base Proxy-oriented data uploading and remote data Integrity Checking (IDPUIC) scheme[2] are implemented using Java language with CloudSim simulator environment. The Amazon EC2 dataset is used for implementation of proposed CTH-CSIV method and existing methods. Amazon EC2 dataset is a cloud computing web service provided by Amazon Web Services (AWS). Amazon EC2 dataset provides data storage using CloudSim simulator with the available resources. After storing the data to a cloud server, the cloud user verifies the integrity of that data. Amazon EC2 dataset information is taken from [21].

The performance evaluation of CTH-CSIV Method is compared with existing methods such as RDIC scheme [1] and IDPUIC scheme [2] with the certain parameters such as space complexity, processing time and data integrity rate with respect to a number of cloud user data.

V. RESULT ANALYSIS

Result analysis of the CTH-CSIV method is discussed in this section. The experimental results of proposed and existing methods are compared using parameters such as space complexity, processing time, and data integrity rate with aid of tables and graph values.

5.1 Impact of space complexity

Space complexity is defined as an amount of storage space required to store the cloud user data in the tree. The space complexity is calculated using a mathematical formula,

$$SC = N * Space\ (storing\ single\ data) \quad (6)$$

From (6),SC denotes a space complexity, N represents a number of cloud user data. It is measured in terms of mega bytes (MB).

Sample mathematical calculation for space complexity

Proposed CTH-CSIV: number of cloud user data is 10 and space utilized for storing single cloud user data is 4.6MB, then

$$SC = 10 * 4.6MB = 46MB$$

ID-based RDIC scheme: number of cloud user data is 10 and space for employed storing single cloud user data is 5.2 MB, then

$$SC = 10 * 5.2MB = 52MB$$

 DOI : 10.18231/2454-9150.2018.0735

IDPUIC scheme: number of cloud user data is 10 and space taken for storing single cloud user data is 5.8MB, then

$$SC = 10 * 5.8MB = 58MB$$

Table 1 Tabulation for Space complexity

No. of Cloud user Data	Space complexity (MB)		
	CTH-CSIV	ID-based RDIC scheme	IDPUIC scheme
10	46	52	58
20	52	62	72
30	63	75	84
40	64	76	88
50	75	85	95
60	78	90	96
70	77	91	105
80	80	96	112
90	88	104	126
100	93	110	130

Table 1 describes the experimental results of space complexity with respect to a number of cloud user data. For the experimental consideration, the number of cloud user data is taken from 10 to 100. The performance of space complexity is analyzed with three different methods of CTH-CSIV, ID-based RDIC scheme [1] and IDPUIC scheme [2]. From the table value, the space complexity using CTH-CSIV method is lesser when compared to existing methods. Based on the above results, the graph is drawn in below figure 6.

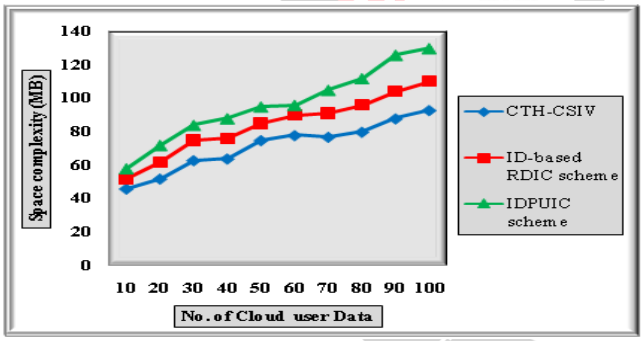

Figure 6 Performance results of Space complexity

Figure 6 depicts the performance results of space complexity versus a number of cloud user data. The cloud user stores their data in the cloud server in a secured manner. The secured cloud storage is achieved by verifying the data integrity. Experimental performances of three different methods are illustrated in Figure 6. The number of user data is taken as input for evaluating the space complexity involved during secured cloud storage. The experimental results reported that the CTH-CSIV method utilizes the minimum storage space for storing the multiple user data in a cloud server. This significant improvement of CTH-CSIV method is achieved by using Concurrent Trie Hashing technique. The Concurrent Trie is used for efficient data storage. Concurrent Trie constructs the tree with a number of nodes such as root node, branch node and leaf node. Each node has a key-value pair for identifying the location of data which is stored. Concurrent Trie stores the hash value of that data instead of storing the original data. Since the original data consumes more space in the cloud storage. In order to overcome this problem, CTH-CSIV method initially generates the hash value for each user data. Then the hashed data are stored in the tree using bit array. In addition, the unwanted data are removed from the tree that minimizes the storage space. As a result, CTH-CSIV method utilizes the less space for storing the multiple user data on a cloud server.

Let us consider the ten different runs to perform the experimental evaluation. After performing the ten runs, the proposed values are compared with existing methods. Then the average is taken for the comparison results. The average value shows that the CTH-CSIV method effectively minimizes the space complexity of cloud data storage by 15% and 25% when compared to existing ID-based RDIC scheme [1] and IDPUIC scheme [2] respectively.

5.2 Impact of processing time

Processing time is defined as an amount of time required for storing the user data in a cloud server. The formula for calculating the processing time is expressed as follows,

$$PT = N * PT\ (storing\ the\ single\ data) \quad (7)$$

From (7), PT denotes a processing time, N denotes a number of cloud user data. PT is measured in terms of milliseconds (ms).

Sample mathematical calculation for processing time

Proposed CTH-CSIV: Number of cloud user data is 10 and processing time required for single cloud user data is 1.8 ms, then $PT = 10 * 1.8 = 18ms$

ID-based RDIC scheme: Number of cloud user data is 10 and processing time taken for single cloud user data is 2.2ms, then $PT = 10 * 2.2ms = 22ms$

IDPUIC scheme: No. of cloud user data is 10 and processing time utilized for single cloud user data is 2.6ms, then $PT = 10 * 2.6ms = 26ms$

Table 2 Tabulation for processing time

IDPUIC scheme	processing time (ms)		
	CTH-CSIV	**ID-based RDIC scheme**	**IDPUIC scheme**
10	**18**	**22**	**26**
20	**24**	**30**	**34**
30	**30**	**36**	**42**
40	**32**	**44**	**52**
50	**40**	**45**	**55**
60	**42**	**54**	**66**
70	**46**	**50**	**63**
80	**44**	**58**	**66**
90	**52**	**64**	**72**
100	**61**	**68**	**76**

Table 2 shows the experimental results of processing time for storing the user data to the cloud server using three different methods namely CTH-CSIV, ID-based RDIC

 DOI : 10.18231/2454-9150.2018.0735

scheme [1] and IDPUIC scheme [2]. The table value clearly shows that the processing time using proposed CTH-CSIV method is minimized when compared to other existing ID-based RDIC scheme [1] and IDPUIC scheme [2]. The graphical results of processing time are shown in figure 7.

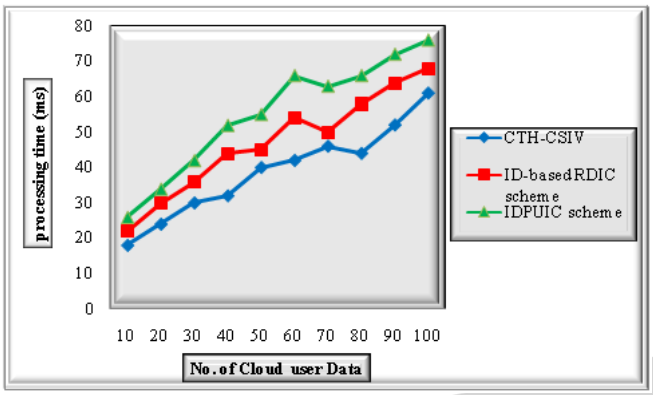

Figure 7 Performance results of processing time

Figure 7 illustrates the performance results of processing time with respect to number of cloud user data. The number of cloud user data is taken for experimental evaluation is varied from 10 to 100. The input data is taken in 'X' direction whereas the performance results of processing time are obtained in 'Y' direction. In figure 7, three different colors of the curve indicate the performance results of three different methods of CTH-CSIV, ID-based RDIC scheme [1] and IDPUIC scheme [2]. The processing time using proposed CTH-CSIV method is minimized as compared to other existing methods. This significant improvement is achieved by applying the concurrent Trie hashing-based data storage. Concurrent Trie hashing stores the number of user data in the cloud server by generating the hash value. The hash values of the data are stored in a bit array. Besides, with the help of hash function, CTH-CSIV method efficiently stores the user requested data in a tree. As a result, CTH-CSIV method securely stores the multiple user data in a cloud server with minimum time. The average results show that the CTH-CSIV method effectively minimizes the processing time of secured cloud data storage by 18% and 30% when compared to existing ID-based RDIC scheme [1] and IDPUIC scheme [2] respectively.

1.3 Impact of data integrity rate

Data integrity rate is measured as the ratio of a number of cloud user data is stored without any modifications to the total number of cloud user data. The data integrity rate is evaluated using following mathematical formula,

$$DIR = \frac{no.of\ cloud\ user\ data\ obtained\ without\ any\ modification}{N} * 100 \quad (8)$$

From equation (8), DIR denotes a data security rate and 'N' denotes number of cloud user data. Data integrity rate is measured in terms of percentage (%).

Sample mathematical calculation for data integrity rate

Proposed CTH-CSIV: Number of cloud user data obtained without any modification is 8, the total number of cloud user data is 10, then $DIR = \frac{8}{10} * 100 = 80\%$

ID-based RDIC scheme: Number of cloud user data received without any variation is 7, total number of cloud user data is 10, then $DIR = \frac{7}{10} * 100 = 70\%$

IDPUIC scheme: Number of cloud user data acquired without any changes is 6, total number of cloud user data is 10, then $DIR = \frac{6}{10} * 100 = 60\%$

Table 3 Tabulation for data integrity rate

No. of Cloud user Data	Data integrity rate (%)		
	CTH-CSIV	**ID-based RDIC scheme**	**IDPUIC scheme**
10	80	70	60
20	90	80	65
30	83	67	57
40	93	83	73
50	86	78	68
60	93	83	75
70	96	89	84
80	94	90	83
90	92	88	81
100	95	89	85

Table 3 shows the performance results of data integrity rate using three different methods namely CTH-CSIV, ID-based RDIC scheme [1] and IDPUIC scheme [2]. After the cloud storage, the integrity of the data is verified. The experimental result of data integrity rate is improved using CTH-CSIV method when compared to existing methods. Let us consider the 10 user data for storing the cloud server, 8 user data are obtained without any modification. But the existing ID-based RDIC scheme [1] and IDPUIC scheme [2] obtained 7 and 6 data without any variations. As a result, data integrity rate of CTH-CSIV method is high when compared to the existing method. The comparison results of three different methods are illustrated in figure 8.

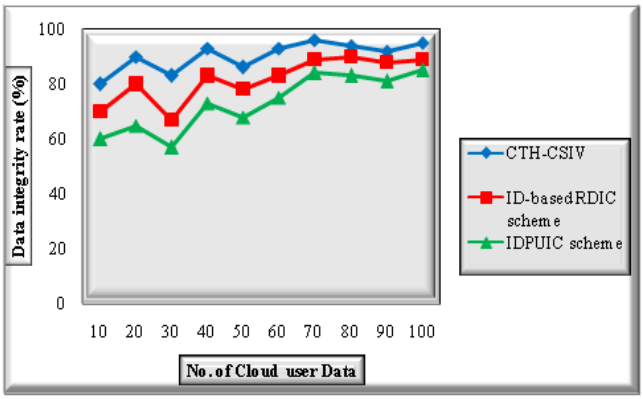

Figure 8 Performance results of data integrity rate

Figure 8 describes the experimental results of data integrity rate with respect to a number of cloud user data. The performance result of data integrity rate is compared with three methods CTH-CSIV, ID-based RDIC scheme [1] and IDPUIC scheme [2]. For measuring the data integrity rate, the numbers of cloud user data are taken as input and it

 DOI : 10.18231/2454-9150.2018.0735

varied from 10 to 100. The integrity rate of CTH-CSIV method is comparatively improved. This higher integrity of CTH-CSIV method is achieved through the integrity verification.

Initially, the cloud user generates the hash value for each data. After generating the hash value, the cloud user generates the checksum. The checksum of hashed data is kept secret by the cloud user. Followed by, the data with the hash value are stored in the cloud server using concurrent trie hashing process. Whenever the cloud user checks their data integrity, then the checksum is again generated for that data. The cloud user performs a verification process to identify whether the data is modified or not. The newly generated checksum signature is accurately matched with the checksum which is already generated. As a result, user data are not altered and obtained high integrity. The above discussion clearly shows that the CTH-CSIV method improves the data integrity in a cloud environment. After performing the ten runs, the data integrity rate of the CTH-CSIV method is significantly improved by 11% and 25% when compared to existing ID-based RDIC scheme [1] and IDPUIC scheme [2] respectively.

VI. CONCLUSION

An efficient CTH-CSIV method is designed with goal of improving the data integrity and minimizing the space complexity of secured cloud storage. The goal of CTH-CSIV method is achieved with help of concurrent trie hashing tree structure and checksum signature integrity verification. With support of concurrent trie hashing tree structure, CTH-CSIV method stores user data with minimal amount of space utilization by constructing a hash value as compared to existing works. Besides with the algorithmic processes of concurrent trie hashing, CTH-CSIV method securely stores the multiple user data in a cloud server with minimal processing time as compared to conventional works. Furthermore with aid of checksum signatures, CTH-CSIV method enhances the verification performance of data integrity in cloud environment as compared existing works. Thus, CTH-CSIV method attains higher security, integrity, confidentiality level on cloud data storage. Experimental evaluation of proposed CTH-CSIV method and existing schemes are carried out using Amazon EC2 dataset. The experiential results of CTH-CSIV method improves data integrity rate by 18 % and also minimizes space complexity of secured cloud data storage by 20 % as compared to state-of-the-art works.

REFERENCES

[1] Yong Yu, Man Ho Au, Giuseppe Ateniese, Xinyi Huang, Willy Susilo, Yuanshun Dai, and Geyong Min, “Identity-Based Remote Data Integrity Checking With Perfect Data Privacy Preserving for Cloud Storage”, IEEE Transactions on Information Forensics and Security, Volume 12, Issue 4, April 2017, Pages 767-778

[2] Huaqun Wang, Debiao He and Shaohua Tang, “Identity-Based Proxy-Oriented Data Uploading and Remote Data Integrity Checking in Public Cloud”, IEEE Transactions on Information Forensics and Security, Volume 11, Issue 6, June 2016, Pages 1165 – 1176

[3] NesrineKaaniche and MarylineLaurent, “Data security and privacy preservation in cloud storage environments based on cryptographic mechanisms”, Computer Communications, Elsevier, Volume 111, 2017, Pages 120-141

[4] HuiCuia, Robert H.Deng, Yingjiu Li, “Attribute-based cloud storage with secure provenance over encrypted data”, Future Generation Computer Systems, Elsevier, Volume 79, Part 2, 2018, Pages 461-472

[5] Shreeraghav Kulkarni and Sujata Terdal, “Ttpse-Trusted Third Party With Symmetric Encryption Towards Secured Cloud Storage”, International Journal of Computer Sciences and Engineering, Volume 5, Issue 5, 2017, Pages 47-51

[6] Dharavath Ramesh, Rahul Mishra, Damodar Reddy Edla, “Secure Data Storage in Cloud: An e-Stream Cipher-Based Secure and Dynamic Updation Policy”, Arabian Journal for Science and Engineering, Springer, Volume 42, Issue 2, 2017, Pages 873–883

[7] Peng Xu, Hongwu Chen, Deqing Zou, Hai Jin, “Fine-grained and heterogeneous proxy re-encryption for secure cloud storage”, Chinese Science Bulletin, Springer, Volume 59, Issue 32, 2014, Pages 4201–4209

[8] Chuan Yao, Li Xu, Xinyi Huang, Joseph K. Liu, “A secure remote data integrity checking cloud storage system from threshold encryption”, Journal of Ambient Intelligence and Humanized Computing, Springer, Volume 5, Issue 6, 2014, Pages 857–865

[9] Lingwei Song, Dawei Zhao, Xuebing Chen, Chenlei Cao, and Xinxin Niu, “A Secure and Effective Anonymous Integrity Checking Protocol for Data Storage in Multicloud”, Mathematical Problems in Engineering, Hindawi Publishing Corporation, Volume 2015, Article December 2014, Pages 1-8

[10] Yunxue Yan, Lei Wu ,Ge Gao, Hao Wang, Wenyu Xu, “A dynamic integrity verification scheme of cloud storage data based on lattice and Bloom filter”, Journal of Information Security and Applications, Elsevier, Volume 39, 2018, Pages 10-18

[11] Laicheng Cao, Wenwen He, Yufei Liu, Xian Guo, Tao Feng, “An integrity verification scheme of completeness and zero-knowledge for multi-Cloud storage”, International journal of communication system, wiley online library, Volume 30, Issue 16 , 2017 , Pages 1-10

[12] Luca Ferretti, Mirco Marchetti, Mauro Andreolini, Michele Colajanni , “A symmetric cryptographic scheme

 DOI : 10.18231/2454-9150.2018.0735

for data integrity verification in cloud databases", Information Sciences, Elsevier Volume 422, 2018, Pages 497–515

[13] Yong Yu, Man Ho Au, Yi Mu, Shaohua Tang, Jian Ren, Willy Susilo, Liju Dong, "Enhanced privacy of a remote data integrity-checking protocol for secure cloud storage", International Journal of Information Security, Springer, Volume 14, Issue 4, 2015, Pages 307–318

[14] Muhammad Imran , Helmut Hlavacs , Inam Ul Haq, Bilal Jan, Fakhri Alam Khan, Awais Ahmad, "Provenance based data integrity checking and verification in cloud environments", PLoS ONE, Volume 12, Issue 5, 2017, Pages 1-19

[15] Liu Hong qing and Huang Yan, "Fault-tolerant Mechanism for Cloud Storage System with Integrity Verification Mechanism", International Journal of Security and Its Applications, Volume 10, Issue 4, 2016, Pages 155-166

[16] Kan Yang and Xiaohua Jia, "An Efficient and Secure Dynamic Auditing Protocol for Data Storage in Cloud Computing", IEEE Transactions on Parallel and Distributed Systems ,Volume 24, Issue 9, 2013, Pages 1717 – 1726

[17] Mohammad Iftekhar Husain, Steven Y.Ko, Steve Uurtamo, Atri Rudra, Ramalingam Sridhar, "Bidirectional data verification for cloud storage", Journal of Network and Computer Applications, Elsevier, Volume 45, 2014, Pages 96-107

[18] Swapnil V.Khedkar , A.D.Gawande, "Data Partitioning Technique to Improve Cloud Data Storage Security", International Journal of Computer Science and Information Technologies, Volume 5, Issue 3, 2014, Pages 3347-3350,

[19] Yuan Zhang , Chunxiang Xu ,Hongwei Li , Xiaohui Liang, "Cryptographic Public Verification of Data Integrity for Cloud Storage Systems", IEEE Cloud Computing ,Volume 3, Issue 5, 2016, Pages 44 – 52

[20] Hongwei Liu, Peng Zhang, Jun Liu, "Public Data Integrity Verification for Secure Cloud Storage", Journal of Networks, Volume 8, Issue 2, 2013, Pages 373-380

[21] Amazon EC2 dataset: http://www.ec2instances.info/

 DOI : 10.18231/2454-9150.2018.0735