An Effectual Homomorphic Tag Based Block for Dynamic Provable Data Possession Framework Based on Block Tagging the Cloud File

K.Kavitha, M.Punithavalli

ABSTRACT--- Storing the data in cloud helps in satisfying the demand of data access at anyplace, anytime. In cloud storage, users authenticate whether the data has been stored to the cloud storage server correctly. In order to enhance the storage provision, an Effectual Homomorphic Tag based Block for Dynamic Provable Data Possession (EHTB-DPDP) framework has been designed. This framework checks for the data integrity in the cloud storage server. The existing PDP, DPDP schemes were analyzed and the drawbacks encountered in those systems have been reframed using the proposed methodology. The major benefit of the proposed EHTB-DPDP is that it offers an effectual dynamic provable data possession and data integrity. This scheme spotlights on the integrity of the remote data by reducing data storage space, so that users can retrieve data efficiently. This security enhancement is achieved by the block tagging methodology. In addition, converting the variable block size to the fixed block size using hash function is also investigated. The feasibility of the scheme is proved by aanlyzing the security and the performance.

Index Terms — Cloud Storage, Effectual Homomorphic Tag based block for Dynamic Provable data possession Possession, Homomorphic Hash Function, Data Possession, Tag block

I. INTRODUCTION

Cloud Computing approach is an on-demand service hosted over the network servers to process, store and to manage the data, rather than personal computer or local server [1]. Cloud services and the applications related to it runs on the distributed network which provides virtual resources to the end users. These kinds of resources can be accessed using various network protocols and standard internet services. Authenticating the data using verification process is a major concern related to cloud data. The main cause for checking data authenticity is extremely high probable malicious activity suffered by both Cloud Service Provider (CSP) and cloud users. There are numerous ways to address this crisis. Cloud users can make use of encryption and decryption process to address this issue. But it requires numerous computational overhead and functional complexities. So data auditing is another way to handle this issue.

Classical ways of auditing the data are Provable Data Possession (PDP) methods [2]-[3]. But there are numerous complexities associated with PDP methods such as computational complexity, storage complexity. These kinds of method s allow only the encrypted data and access only limited amount of queries. The PDP model is not appropriate for batch auditing because of the computational complexity [4]-[5]. The advantages of these methods are only the pre-processing stage that can be applied for the outsourced data. As well, these methods do not preserve privacy. There is a trade-off between storage overhead and communication cost. Some PDP necessitates higher cost for lesser storage. In order to overcome the shortcomings of PDP [20], Dynamic Provable Data Possession (DPDP) was proposed [6].

Dynamic Provable Data Possession (DPDP) provides a strong model to offer guaranteed data integrity by extending the dynamic range of operations over the outsourced data, such as insertion, modification, deletion and append [8]. DPDP protocol comprises of three stages for static data (Setup, Challenge, and Retrieve) along with the update phase. In setup phase, homomorphic algorithm will be used to encrypt the data. In the updation phase, the original file may be updated. During challenge phase, the data integrity is achieved by the latest version of updated file (it may be a different file). In retrieve phase, the client retrieves the updated version of the file. The foremost challenge with handling the static data in DPDP [7] is to ensure that the client has received the latest file version (i.e. preventing the use of old file) while fulfilling the overhead requirements. DPDP framework is an amalgamation of diverse polynomial time algorithms (KeyGen DPDP, PrepareUpdate DPDP, PerformUpdate DPDP, VerifyUpdate DPDP, GenChallenge DPDP, Prove DPDP, execute DPDP). With the use of seven polynomial algorithms DPDP fails to provide robustness in the cloud service. In order to overcome the shortcomings of the DPDP scheme, the proposed work has anticipated a new framework known as Effectual Homomorphic Tag Based Block for Dynamic Provable Data Possession (EHTB-DPDP). Figure 2 shows the pre-processing and verification of data in cloud to enhance data integrity [2].

Revised Manuscript Received on July 10, 2019.

K.Kavitha, Assistant Professor, Department of Computer Science, Dr.G.R.Damodaran College of Science, Coimbatore, T.N, India.

Dr.M.Punithavalli, Professor, Department of Computer Applications, Bharathiar University, Coimbatore. T.N, India.

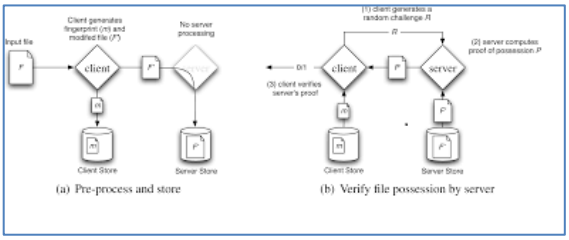

Figure 1: Pre-processing and verification for data integrity in cloud representation

EHTB-DPDP posses tagging the file stored in the cloud, in addition the variable block size is also converted to fixed block size using hashing function. Here, BlockTag algorithm has been designed to improve the authenticity of the stored file in the cloud. The design goal of the proposed EHTB-DPDP protocol is summarized below:

1) To effectively and securely allow authorized users to access the file from CSP.
2) To use a set of tags for the files in the cloud storage for verification purposes.
3) To enable data owner to execute dynamic data updation process while maintaining the consistent level of data.
4) To ensure data owner with the evidence that CSP possesses all the copies of data.

The rest of the work is organized as follows: Section II explains the related work based on PDP methods and homomorphic encryption. Section III elaborates the proposed EHTB-DPDP framework for blocks and tag generation along with the additional parameters evaluation. Section IV shows the experimental analysis and the results associated with the proposed work. Section V describes about the security and the additional storage acquired on executing the proposed work. Section VI concludes and discuss the future work.

II. RELATED WORKS

Rajat Saxena, [2016] anticipates enhanced data integrity verification methodology with the use of multiple third party auditors. This approach makes use of Paillier Homomorphic Cryptography [PHC], Combinatorial batch codes [CBC] and homomorphic tag for the purpose of data integrity verification. This approach is appropriate for cloud storage as the homomorphic tag efficiency and with the PHC advantages. Moreover, this approach satisfies dynamic data operation with reduced overhead. Here, CSP does not require added data structure to organize data operations. It offers enhanced security during Traffic flow examination, Man in the Middle attack (MITM), Defacement, Impersonation and data storage misuse due to the Pailliers self binding property, which has the ability to change cipher text without any alterations in plain text and intruders misguide. Finally, the recital of this approach is not bounded with disk I/O, in which the comparison with the prevailing methods shows usefulness and effectiveness of the method.

Ertem Esiner, [2014] described a novel data structure (FlexList) and its optimized implementation in cloud data storage. FlexList efficiently assists variable block sized dynamic provable updates and this approach assists in handling multiple updates and proofs at considerably by enhancing scalability. Energy efficiency of FlexListbased and FlexDPDP was studied for cloud storage. This method also illustrates how to build data structure from scratch with O (n) time, indeed of O(n log n) time. This method anticipates how to parallelize this kind of authenticated structure.

Ertem Esiner, [2014] extended FlexDPDP process with the use of efficient and optimized algorithms, and examine their recital with the real world network realistic settings. The speed obtained using this method is 6 while using 8 cores of pre-processing phase, 60% enhancement on updates in server side and 90% enhancement in checking the client side. This method was deployed on PlanetLab testbed and offer detailed examination using real version workload traces control system.

K.Renugha, [2017] described about exclusive-or homomorphism encryption process is executed on protecting data searching method. The new system initiates randomization process for every session as data pattern can be conserved. Searching examination can be carried out by on-demand calculation grounded on session key generation by randomization technique and require not store key in cloud. Hashing based indexing is cast of to enhance searching performance. This method is verified experimentally by searching files in untrusted server environment. XOR homomorphism encryption process is practically executed and it proves that this scheme is extremely efficient.

Junyao YE, [2016] anticipates PDP based homomorphic hash function in accordance to the problems encountered in various literatures. This technique permits users to guarantee data integrity in server for unlimited times of iteration. It also offers provable data possession in data integrity protection and server. Users merely require save parameters, transmission data is little in verification procedure, and provable data possession verification is one time homomorphic hash computation. Security examination and performance examination prove that this technique is feasible. The method obtains data recovery. Here, error-correcting codes or erasure codes are utilized to encode data before computing hash value.

Ayad F. Barsoum, [2015] deliberates a novel PDP scheme (known as MB-PMDDP), which assists in outsourcing of multi-copy dynamic information, where data owner is competent of not accessing and archiving data copies stored by CSP, but as well scaling and updating the copies on remote servers. The proposed method is to address multiple copies of dynamic data. Interaction amongst CSP and authorized users is measured in this process, where authorized users can flawlessly make use of data copy obtained from CSP by single secret key shared amongst data owner. Furthermore, proposed scheme assists public verifiability, permits possession-free verification and arbitrary number of auditing where verifier has competencies to check data integrity though possesses or retrieves file blocks from server.

Yasmina Bensitel, [2016] describes the utilization of fully homomorphic, and illustrates that this method does not provide best solution. Hybrid partial homomorphic encryptions and somewhat homomorphic encryptions can be cast of indeed of the existing methods. Some instances are provided for certain statistical functions utilized in real life for medical application, and which is utilized over encrypted data.

Adil Bouti, [2015] presented protocol improvement for evaluation on encrypted data in clouds. The overhead is measured to be low to make the protocol in implementation. Distribution of computation between numerous cloud providers enhanced security at additional communication cost enhancement to protocol for calculation on encrypted data in clouds.

Clementine Gritti, [2015] provided two solutions to resolve adversarial crisis anticipated in DPDP scheme with DP and PV proposed. These solutions assists in overcoming replay attacks, modifies attacks and attacks over data privacy by bombarding MHT or IHT into construction. Two novel schemes are secure against server and data privacy-preserving against TPA in random oracle.

III. PROPOSED METHODOLOGY

a. Model Design

The system model comprises of three different categories that are shown in figure 3. These categories are extremely significant in the process of cloud storage.

1) **Cloud Service Provider (CSP):** CSP is a third party who offers storage services to data owners/holders. Data owners can upload data to storage space offered by CSP [12]. Auditing also performed by CSP when data owner request for data integrity [13]-[14].
2) **Data owner (D):** D is an individual or an enterprise that outsources data in cloud. 'D' will partition the file of variable block size into fixed sized data blocks and produces multiple data blocks models.
3) **User (U):** U has adequate access rights to use or share data blocks stored in S [15]-[18]. User U will holds valid decryption key to access the entire encrypted data blocks.

In the proposed framework, the files are partitioned into blocks, and generate tags for each block. Then, computes a hashing value for every tag to guarantee tag integrity and use these tags to guarantee the file block integrity. This EHTB-DPDP framework supports key generation, updation, and verify, prove, challenge, prove, execute along with the chunks like setup [insert, delete, modify].

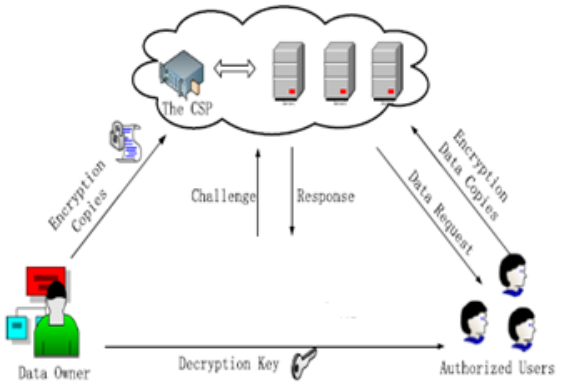

Figure 3: Pictorial representation of the system model.

Subsequently, the building block of homomorphic encryption scheme is discussed in this section. Key generation is the process in the algorithm that provides outputs public key P_k and private key P'_k. Encrypt function is the process that consumes message 'm' and public key P_k and provide outputs ciphertext 'c'. Decrypt function is the process ciphertext 'c' and private key P^1_k and outputs message 'm'. Function evaluation is step in algorithm that considers evaluation key, set of cipher texts $c_1.....c_n$, circuit, and ciphertext c_e outputs. Circuit specifies certain function realized with logical gates.

A list will be maintained to search the specific block of the stored file quickly. Assume, 'F' is a file with 'n' blocks $\{x_1, x2, x3,...x_n\}$. The server/client stores the block at the bottom level, which is identified using a pointer. Hence, the server can easily identify the block of the file in the list using the pointer quickly.

The client generates an array 'β'. The item 'V_i' corresponds to the block X_i signifies the number of times the block has been modified/updated. The client computes tag 'T' to every block along with the hash value for the corresponding block. Pointer has been maintained amongst the blocks, tags, list, array and the hash value.

b. EHTB-DPDP FRAMEWORK

i) Description

The ultimate purpose of the proposed method EHTB-DPDP framework is to permit the users to check whether untrusted storage server maintains the data appropriately. Usually, there are two parties in cloud: storage server and client. The scheme of the proposed methods comprises of various phases based on homomorphic hash function 1) Setup; (2)TagBlock; (3)Challenge; (4) ProofGen; (5) ProofVerify (6)Execute (7) Update.

Firstly, we need to divide the file 'F' into 'n' blocks. In the following phases such as TagBlock phase and ProofVerify phase, all the calculations are based on the file blocks.

ii) Design Goal

The proposed scheme should fulfil the properties:

(1) High efficiency: to permit data owner to resourcefully verify integrity of numerous data copies [21]-[22].

(2) To guarantee that cloud servers should not cheat users [19] if there are no copies of data.
(3) Assist dynamic operations: to permit data owners to regularly update outsourced data by doing insert, modify, delete, and append operations simultaneously.

iii) Keygen phase

The client makes use of a key generation function *Keygen* generate private key and public key { $P'_{k,}$ P_k}. The public key is transferred to the server while the private key is maintained secretly. Partition the files to variable block { $x_1,x_2,x_3,...x_n$}, call the *BlockTag* to generate tags to every block. Hashing functions are used to convert the variable block size to fixed block size and the array β is build at the client-side, finally every item is initialized to 0 initially with the pointer. Figure 4 shows the items to be stored in the list of hash array.

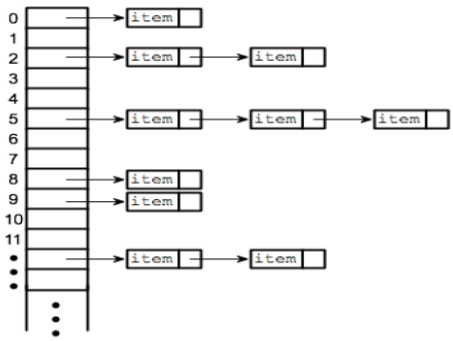

Figure 4: Hash array for storing the block of file in cloud

iv) KeyGen (1^k) → { $P'_{k,}$ P_k}

1. Initialize $\mathbf{1^k}$
2. Evaluate $P'_k = k_1$, $K_1 \leftarrow \{0,1\}^k$; $P_k = (M, g)$, M= pq is the product of two prime and the g is the high order in Z^*_N
3. Output { $P'_{k,}$ P_k}

v) BlockTag ($P'_{k,}$ $P_{k,}$ $X_{n,}$ V_i, i) → {T_i, h_i}

1. Initialize $P'_k = (N,g)$, $P'_k = k_{1,}$ block file X_i, block index 'b'
2. Compute $T_{i\,=}\, g^{xi}$ mod N, $h_i = H_{k1}$ (T_i || $F(v_i)$ || i), H is a hashing function, F is a pseudo random function
3. Output {T_i, h_i}

The combination of block tags, block set, hashing value and the list constitute the file processed. The client uploads the file processed and the public key to the server, thus maintains the array and private key secretly.

vi) File update → Insert

Update→ insert, inserting a new block X_i^* after the existing block X. Client adds V^*_{n+1} at the end of β array (considering the 'X' original block), that is initialized to 0, generate a tag T_i^* and hashing value h_i^* for X_i^*, send (T_i^*, h_i^*, X_i^*) to server. The server changes the position of the pointer, and inserts the new tag T_i^*, new hashing value h_i^* and the new block X_i^*, finally maintains the pointer amongst the list. The client sends the hashing function H_k and randomly generates g^s to server to check whether the update→ insert is performed successfully.

1. Client: adds V^*_{n+1} at array end β, that has been initialized to 0, compute $T_i^* = g^{m*}_i$, $h^*_{n+1} = H_{k1}$ (T^*_i || F (V^*_{n+1}) || n+1), sends (X_i^*, T_i^*, h^*_{n+1}) to server.
2. Server: Move the pointer to the bottom level of the list, insert X_i^*,T_i^* at the appropriate position, add h^*_{n+1} at hashing value array end, maintain the pointer and update the list.
3. Client: selects hashing function H_k and random number S, compute $g_s = g^s$, send to server.
4. Server: search list to find $(j+1)^{th}$ block, and its corresponding hashing value $h_{(j+1)}$, tag T_{j+1}.
5. Compute $T_s = g_s^{mj+1}$ mod N, h= $H_k(T_{j+1}$ || $h_{(j+1)}^{/})$, sends (T_s,h) to clients.
6. Client: Compute $(Ti^*)^s$? =T_s,H_k (T_i^* || h^*_{n+1})? = h.

The update → insert is successful.

vii) File update → Edit

File update → Edit, edit the i^{th} block X_i to X_i^*. The index i is transferred to server. Server searches the list to identify the bottom level and its subsequent block X_i, hashing value h_i, tag T_i. The client checks the data integrity by hashing value, block, tag and the corresponding β array. Client updates the item V_i to V_i^* in β array, updates X_i to X_i^*, T_i to T_i^*, h_i to h_i^* and send it to the server. Server updates the corresponding hash value H_k and randomly generated g^s to check whether the update→ edit operation is successful.

1. Client: transfer i to server
2. Server: search the list to find block m_i, tag T_i, hash value $h_i^{/}$ and the index $i^{/}$ to client
3. Client: Evaluates $T_i^{?}=g^{Xi}$ mod N, $h_i?=H_{k1}(T_i$ || $F(V_i^{/})$ || $i^{/})$ to check the block integrity, update $v_i^{/}$ in β array , update $m_i \rightarrow m_i^*$, computes $T_i^* = g^{X*}_i$ mod N
4. H_i*= H_{k1} (T_i || F (V_i^*) || $i^{/}$, sends (X_i^*, T_i^*, h_i^*) to server
5. Server: Update $m_i \rightarrow m_i^*$, $T_i \rightarrow T_i^*$, $h_i \rightarrow h_i^*$
6. Client: sends hashing function H_k and random number $g_s=g^s$ to server
7. Server: Searches list to find the block X_i, and its tag T_i, subsequent hash value $h_i^{/}$, computes $T_s=g_s^{Xi}$ mod n, h = $H_k(T_i$ || $h_i^{/})$, sends (T_s,h) to client
8. Client: Evaluate $(T_i^*)^s$?= T_s, $H_k(T_i^*$ || $h_i^*)$?= h.
   The update→ edit is successful.

viii) File update → delete

File update→ delete, delete any block from the server storage. Client transfers the index to the server. Server searches the list to find block X_i, hashing value h_i, tag T_i, finally it deletes the block and the corresponding tags, hashing value and index from the server. Client updates it and sends it to server as shown in figure 4. Client send the hashing function H_k and the randomly generated g^s to server, to check whether the delete operation is performed successful.

1. Client sends the item i to be deleted to the server
2. Server: Search the list to find the block X_i, Tag T_i, hashing value $h_i^{/}$ and deletes X_i and T_i and the hashing value related to it and updates the list to client
3. Client verifies the correspondence between hashing value and the tag, the pointer changes its position.

4. Server: updates $h_i^{/} \rightarrow h_i^{*}$, delete h_n
5. Client: selects hashing function and random number s, $g_s=g^s$, sends to server
6. Server searches the array to find the hash value, blocks and tags. $T_s = g_s^{X^*_i}$, Mod N, h= $H_k(T_i \parallel h_i^{/})$ sends to client
7. Compute $(T_n^{/})^s$=? T_s, $H_k(T_n^{/} \parallel h_i^{*})$? = h
   The update delete is successful.

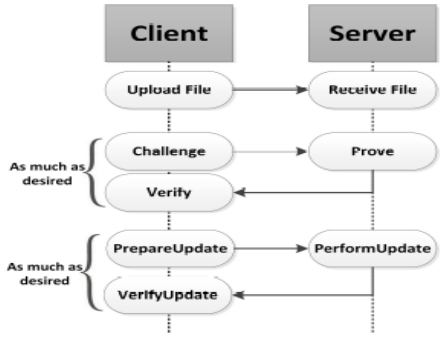

Figure 5: Flow of data upload, update and verification process in cloud

ix) $(F^{/}, \varphi\_) \leftarrow ExecUpdate\ (F, \varphi, Update)$

This algorithm is executed by CSP, in which the input parameters are files 'F', tags 'T' and request for updation (sent by data owner). The output provided will be the updated version of the file, '$F^{/}$' along with the updated signatures 'φ'. After performing any changes in the file block, data owner executes the challenge protocol, to guarantee that the operation performed by the cloud is absolutely correct. The update operation may be inserting a new file or deleting a file or it may be modifying the file.

x) P←Prove (F, φ, challenge)

This algorithm is run by the CSP. It takes the replicas of file F, the tags φ and challenge vector sent by the data owner as input and returns a proof P which guarantees that the CSP is actually storing s copies of the file F and all these copies are intact. The data owner uses the proof P to verify the data integrity. There are two phases in this algorithm:

This process is carried out by the CSP. It generates the file replica, tags 'T' and challenge sent provided by the data owner as input and returns proof 'P' to ensure that CSP is storing the file copies F and these copies are actually intact. Hence, the data owner makes use of 'P' to check the data integrity. There exists two processes in this phase, one is challenge phase and subsequent one is response phase.

a) Challenge: Here, data owner confronts to check for integrity for the outsourced copies [23]-[24]. This challenge phase in cooperate two verification schemes: i) Deterministic—all file blocks from acquired from the file copies are utilized for verification ii) Probabilistic—only certain blocks of all the copies were used for verification.

1. Input: Number of blocks to be challenged
2. Select two random numbers
3. Output: challenge = $\{(i_1,...i_c), (a_1,...a_c)\}$

Probabilistic key is utilized to produce random indices ranging from 1 and m. File blocks obtained from these indices are cast off for verification. In each verification process, percentage of file verified and it maintains the account for entire file verification. At challenge phase, data owner selects verification scheme he wishes to use. If owner selects deterministic verification scheme, he generates Key1. If he selects probabilistic scheme he produces two keys, Key1 and Key2. Key1 generates c ($1 \le c \le m$) random file indices which signify file blocks used for verification by the cloud service provider. Key2 generates random values and CSP should use these random numbers for each file copy during computation of response. Data owner transmits the generated keys to CSP.

b) Response: Response phase is carried out by CSP, while challenge for data integrity verification is acquired from data owner. Here, proof for probabilistic verification scheme (deterministic verification follows the same procedure). CSP obtains two keys, Key1 and Key2 from data owner. With Key1, CSP generates set {C}, ($1 \le c \le m$) random file indices ({C} < {1, 2, . . . ,m}), which specifies file blocks that CSP utilized for verification. With Key2, CSP generates random values T = $\{t_1, t_2, . . . ,t_s\}$. Cloud carry out two operations; one on tags and other on file blocks.

i) Tag operation: Cloud multiplies file tags related to file indices produced by Key1.

ii) File block operation: Cloud takes every file copy and multiplies the entire file blocks related to file indices produced by Key1. Product of each copy is raised as power to random number generated for that specific copy by Key2.

1. Input: Query challenge and file server stored
2. Search hash array to obtain corresponding tags and blocks through pointer
3. Output: Block file

If file size enlarges, more blocks are needed to specify the file. In all the cases, the performance of the proposed method remains the same. It does not degrade the performance of computation [25] as that of the traditional methods. This is because, the entire blocks are monitored by the hash table, if there is any overflow in the amount of block obtained, it automatically re-constructs the block storage structure. Even though deletion and insertion operation has complications, it does not cause any raise in computational complexity [26]. The file blocks generated after homomorphic addition signifies the encrypted file blocks of the updated version requested by data owner [28]-[30]. Encrypted file blocks are given to the data owner, in which the data owner decrypts the file block to acquire the requested version.

IV. SECURITY ANALYSIS

Here, the formal analysis of security provided by our proposed method is examined. Initially, data owner encrypts files and store it over the cloud. The cloud is an untrusted medium; hence the cloud is identified as a preliminary adversary in this method. The proposed scheme is secure, when the cloud does not cheat the data user or the owner by modifying the file blocks and as well pass the response/challenge phase generated by data owner. The proposed scheme offers flexibility to data owner to transmit different files and keys in challenge phase to CSP. This

guarantees that response produced by CSP will not be same for all challenge transmitted by data owner [27]. This neglects the opportunity for CSP to forge response devoid of actually computing it.

a. Security against deletion of file blocks with same value

When the files are partitioned into blocks, there is a chance that only few blocks will have the similar values. The data tags will be same for all the blocks with same value. Even though, the file blocks have similar value, their cipher texts will not posses similar values. File blocks are encrypted and those encrypted blocks do not possess similar value. The cloud can only identify the file blocks with similar value by recognizing the tags with same value. Cloud can also over loop data owner by storing one data block and eliminating data blocks that possess same file tag. In order to avoid this over loop, the proposed scheme randomizes data before generating tags. Data in the tags are summed up with the randomly generated numbers from the private key as is termed as keytag. Hence, even if the tag values are same, the data file underlying will not possess same value.

V. RESULTS AND DISCUSSION

To execute the proposed strategy, the simulation of the proposed method is examined under MATLAB environment. The experiments were conducted on the top of private cloud platform with diverse configurations on Windows 7 operating system. Cloud user utilizes the Windows virtual server for storage infrastructure. Based on users' requirements, they can decrease or increase the storage locations. Local cloud is utilized to retrieve or store the data. This can also be examined in the EC2 and Amazon S3 cloud in the future. However, it facilitates the owner to store the copies of files on cloud server which is located in various geographical locations. The computational time for the diverse operations performed by the data owner, CSP and verifier is recorded. Thus, the computational efficiency of the proposed method on different copies is also investigated in the table.

The experiment is performed on a system with Intel(R) Core(TM) 3.10 GHZ processor and 4 GB RAM computer. The Pairing Based Cryptography (PBC) library was utilized to implement cryptographic operations in the mechanism. Encryption security parameters was set up as 'λ' = 60. The copies of data file of size 1, 5, 10 and 20 MB are stored to private cloud correspondingly and their copies are partitioned into 218 blocks. Assume that the desired security level is 128-bit, and thus the proposed work dealt with 256-bit group order. The computational cost for each phase (file size 1 MB) in this protocol is given in Table II.

Table I shows the various conventional PDP schemes' merits and demerits with the proposed method.

Table I: Comparison of the existing PDP scheme with the proposed methodology with the merits and demerits

S.No	Existing PDP scheme	Merits	Demerits
1	PDP	1. Protection during small corruptions. 2. Reduced update block complexity 3. RSA is used for security. 4. Permits public verifiability	1. Block searching is poor 2. It applicable only for static files. 3. It is insecure against dynamic data block.
2	DPDP	1. Block modification and updating is allowed. 2. Integrity verification is efficient due to querying and updating DPDP scenario.	1. Client performs extra computations. 2. Construction of rank based scheme is complex.
3	CPDP	1. Permits multi cloud storage. 2. Hash index hierarchy minimizes search complexity.	1. Due to multi cloud storage, Combiner model needs to be added which may increase complexity.
4	SPDP	1. It offers secure PDP by encryption 2. It is light weight PDP scheme as it facilitates Homomorphic hash function.	1. Fails in randomness. 2. Client can easily deceive the server.
5	EHTB-DPDP	1. Permits multi-cloud storage. 2. Provides more flexibility as block is partitioned in multiple parts. 3. Tagging every file block enhances security. 4. Varied size key generation for every block reduces extraction of data from cloud by unauthorized party. 5. Reduced computational time for generating variable key size.	1. Incurred overhead

a. Computational Complexity

Server side complexity computation: During verification, server computes 'c' hash integers $H(m_i)$, $1<i<c$. Then, it calculates value $K= C_1h(m1)+C_2h(m2)+...+C_ch(m_n)$. The computation of each $c_ih(m_i)$ corresponds to the product of two integers being t and h bits long. Hence, upper bound is obtained on server's computation time:

$$|c|time_{hash}+ |t_c| time_{add} \quad (3)$$

Verifier side complexity computation: Except for additional pseudorandom number generations corresponding to challenge, computing cost analysis 'R' is similar to client side also. Therefore, verifier computation time is upper bounded and given by,

$$|c|time_{prng}(t)+|t_c|time_{add}(t_h)+time_{hash}(r) \quad (4)$$

The computation complexity between any server and verifier is slightly higher, and still very reasonable. This does not incur time for verifier of data blocks to be detected, as computational complexity requires very small mathematical calculation. Table II shows the computing the complexity and the comparison with the existing work. The communication cost related to block size is shown in figure 15.

Table II: Comparison of computational complexity of the proposed Vs existing scheme

Scheme	Server Computation	Client Computation	Communication overhead
PDP	$O(1)$	$O(1)$	$O(1)$
Scalable PDP	$O(1)$	$O(1)$	$O(1)$
DPDP I	$O(\log n)$	$O(\log n)$	$O(\log n)$
DPDP II	$O(n€ \log n)$	$O(\log n)$	$O(\log n)$
EHTB-DPDP	$O(n)$	$O(n)$	$O(n)$

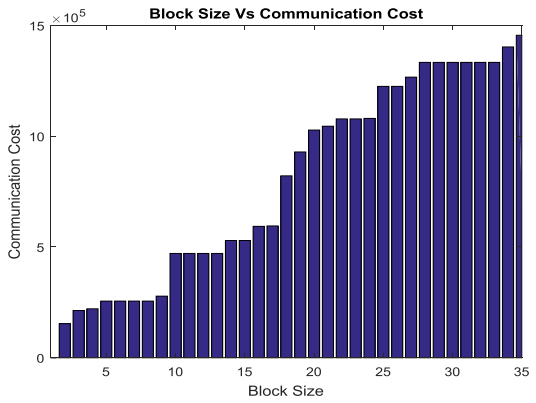

Figure 15: Graphical representation of block size Vs communication cost

b. Additional Storage

One of the parameter to check the efficiency of the proposed method is the calculation of additional storage. Additional storage is the proof for dynamic data storage both at the server and the client side. Client side comprises of private key and the hash array, while the server side comprises of hashing value and the tag set. For instance, assume if as 4GB file 'F' is partitioned into 1,000,000 4KB blocks, and every block comprises of 128B tag and 20B hash value. The hash list has 'n' nodes, and every node is 4B, therefore the additional storage encountered in the server side is approximately about 152 MB, which is the about 4% of original file. The hash array has 1,000,000 times for every item is 2B (216=65536 time). Thereby, the additional storage in client side is 2 MB, which is roughly of 0.06% of original file. Table III shows the time consumed for accessing the file with varied block size.

Table III: Table for computing the data in the block file with the time utilized for execution

Data block size [GB]	Time (ms)
64 (2^6)	7.982
128 (2^7)	20.245
256(2^8)	37.554
512(2^9)	102.167
1024(2^{10})	351.012

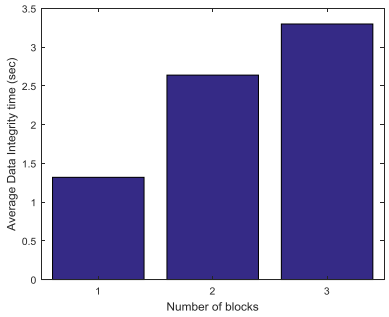

Figure 16: Graphical representation of block size Vs average data integrity time

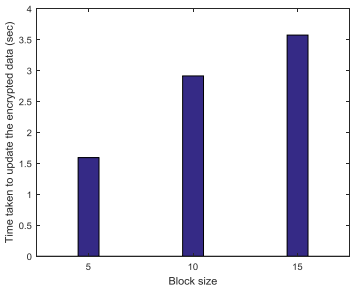

Figure 17: Graphical representation of block size Vs file updating time in seconds

Figure 16 and 17 shows the graphical representation of average data integrity with respect to the number of size and the time taken to update the blocks in the file. Compared to the conventional DPDD scheme, our proposed model shows better storage of data, i.e. 4% at server side and 0.06% at client side. The proposed method shows increased rate of additional storage. This leads to the reduction of communication and computational complexity. As the storage capacity growth is within the range.

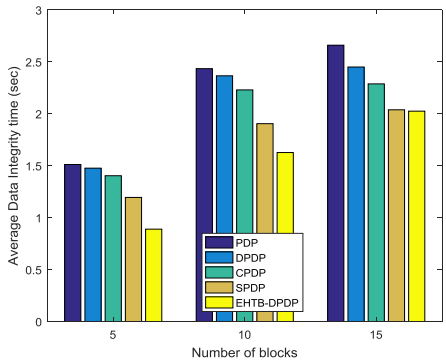

Figure 18: Graphical representation of number of blocks Vs data integrity time in sec

Figure 18 shows the graphical representation of average data integrity time in seconds for the sum of blocks in the cloud environment. The time taken to perform this operation by the proposed EHTB-DPDP method is lesser in contrast to the existing methods such as PDP, DPDP, CPDP, BPDP methods. The proposed EHTB-DPDP method shows better trade off than the prevailing methods in terms of security and performance.

Table IV: Comparison table for computing average data integrity time in seconds of existing and the proposed EHTB-DPDP method

S.No	Various Schemes	Iterations 1	Iterations 2	Iterations 3
1	PDP	1.5113	2.4342	2.6596
2	DPDP	1.4760	2.3652	2.4496
3	CPDP	1.4033	2.2293	2.2873
4	SPDP	1.1942	1.9044	2.0379
5	EHTB-DPDP	0.8889	1.6264	2.0247

Table IV illustrates the average time taken for data integrity in cloud environment. The anticipated EHTB-DPDP method shows lesser time for attaining integrity than the existing methods. Assume if the block size increases from 5, 10, 15 and so on, the time taken by the proposed method is 0.8889, 1. 6264, 2.0247 respectively. It is lesser when compared to the existing methods.

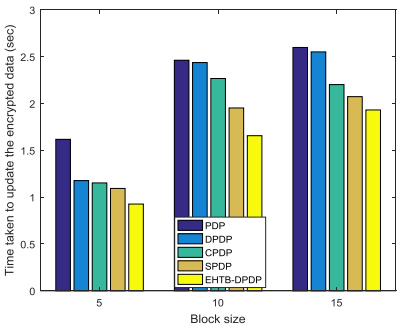

Figure 19: Graphical representation of number of blocks Vs time taken to update the entire block in sec

Figure 19 shows the graphical representation of time taken to update the encrypted data in seconds for the sum of blocks in the cloud environment. This graph considers blocks of three different sizes. The time taken to perform this operation by the anticipated EHTB-DPDP method is lesser in contrast to the existing methods such as PDP, DPDP, CPDP, BPDP methods. The proposed EHTB-DPDP method shows better trade off than the prevailing methods in terms of security and performance.

Table V: Comparison table for updating the encrypted data(secs) of existing and proposed EHTB-DPDP method

S.No	Various Schemes	Iterations 1	Iterations 2	Iterations 3
1	PDP	1.6176	2.4620	2.5984
2	DPDP	1.1764	2.4370	2.5512
3	CPDP	1.1522	2.2669	2.2018
4	SPDP	1.0932	1.9524	2.0730
5	EHTB-DPDP	0.9264	1.6562	1.9311

Table V depicts the updation of encrypted data in cloud environment. The anticipated EHTB-DPDP method shows lesser time for updating the blocks than the existing methods. Assume if the block size increases from 5, 10, 15 and so on, the time taken by the proposed method is 0.9264, 1. 6562, 1.9311 respectively. It is lesser when compared to the existing methods.

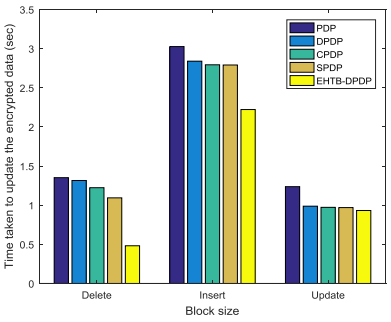

Figure 20: Graphical representation of number of blocks Vs time taken to update the encrypted data in sec

Figure 20 shows the graphical representation of time taken to update the encrypted data in seconds for the sum of blocks to perform insert, delete, and update operation in the cloud environment. This graph considers blocks of three different sizes. The time taken to perform this operation by the anticipated EHTB-DPDP method is lesser in contrast to the existing methods such as PDP, DPDP, CPDP, BPDP methods. The proposed EHTB-DPDP method shows better trade off than the prevailing methods in terms of security and performance.

Table VI: Comparison table for performing insert, delete and update operation of existing and proposed EHTB-DPDP method

S.No	Various Schemes	Iterations 1	Iterations 2	Iterations 3
1	PDP	1.3514	3.0256	1.2365
2	DPDP	1.3151	2.8390	0.9869
3	CPDP	1.2231	2.7929	0.9721
4	SPDP	1.0933	2.7902	0.9690
5	EHTB-DPDP	0.4813	2.2212	0.9315

Table VI shows the time taken to update the encrypted data in cloud environment. The anticipated EHTB-DPDP method shows lesser time for inserting, deleting and updating the blocks than the existing methods. Assume if the block size increases from 5, 10, 15 and so on, the time taken by the proposed method is 0.4813, 2.2212, 0.9315 respectively. It is lesser when compared to the existing methods.

VI. CONCLUSION

This investigation mainly focussed of efficient storage of data in the untrusted cloud server storage. Hence, this research work introduced an Effectual Homomorphic Tag based Block for Dynamic Provable Data Possession (EHTB-DPDP) framework, in which it suitable to minimize the block access by introducing tagging with variable file sized block. This computation is held in both the client and also in the server side. The solution generated by the proposed method fits to reduce the overhead at server side with constant amount of communication. The significance of the proposed work lies in homomorphic verifiable tags. It facilitates data possession devoid of having access to actual data file. Experiments conducted on this scheme, leads to assuring probabilistic possession by sampling server storage, and practical implementation is also carried out in large datasets, whereas the traditional methods fails in achieving proven possession in large datasets. The investigation shows that the scheme imposes significant computational complexity and additional storage on server. Further work can be extended for checking the file integrity and remote procession with the use of proposed methodology.

REFERENCES

1. Feng Dengguo, Zhang Min, Zhang Yan, et al. Study on cloud computing security. Journal of Software, 2011, 2(1):71-83
2. Ateniese G, Burns R, Curtmola R, et al. Provable data possession at untrusted stores. Proc of the 14th ACM Conf on Computer and Communications Security. New York: ACM, 2007: 598-609.
3. Da Xiao, Jiwu Shu, Kang Chen,et al. A practical data possession checking scheme for networked archival storage. Journal of Computer Research and Development, 2009, 46(10):1660-1668.
4. Ateniese G, Dipr, Mangini L V, et al. Scalable and efficient provable data possession. Proc of the 4th International Confon Security and Privacy in Communication Netowrks (SecureComm 2008). New York: ACM, 2008:1-10.
5. Chen B , Curtmola R. Robust dynamic provable data possession. Distributed Computing Systems Workshops (ICDCSW), 32nd International Conference on . Macau: IEEE, 2012: 515-525.
6. Zhu Y, Wang H, Hu Z, et al. Cooperative provable data possession. Beijing: Peking University and Arizona University, 2010.
7. Zhao Kaiyong, Chu Xiaowen, Wang Mea. Speeding up homomorpic Hashing using GPUs. The 2009 (44th) IEEE Conference on Communication (ICC 2009), Dresden, Germany, June 14-18, 2009: 1-5.
8. Erway, C., K¨up¸c¨u, A., Papamanthou, C., Tamassia, R.: Dynamic provable data possession. In: Proceedings of CCS 2009, pp. 213–222 (2009)
9. Esiner, E., K¨up¸c¨u, A., ¨Ozkasap, O.: Analysis and optimization on flexDPDP: a practical solution for dynamic provable data possession. In: Proceedings of ICC 2014 (2014)
10. Gritti, C., Chen, R., Susilo, W., Plantard, P.: Dynamic provable data possession protocols with public verifiability and data privacy (2015).
11. Gritti, C., Susilo, W., Plantard, T.: Efficient dynamic provable data possession with public verifiability and data privacy. In: Foo, E., Stebila, D. (eds.) ACISP 2015. LNCS, vol. 9144, pp. 395–412. Springer, Cham (2015).
12. Wang, B., Li, B., Li, H.: Knox: privacy-preserving auditing for shared data with large groups in the cloud. In: [12] Bao, F., Samarati, P., Zhou, J. (eds.) ACNS 2012. LNCS, vol. 7341, pp. 507–525. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-31284-7 30
13. Wang, B., Li, B., Li, H.: Panda: public auditing for shared data with efficient user revocation in the cloud. IEEE Trans. Serv. Comput. 8(1), 92–106 (2015)
14. Wang, C., Chow, S., Wang, Q., Ren, K., Lou, W.: Privacy-preserving public auditing for secure cloud storage. IEEE Trans. Comput. 62(2), 362–375 (2013)
15. Yu, Y., Au, M.H., Mu, Y., Tang, S., Ren, J., Susilo, W., Dong, L.: Enhanced privacy of a remote data integrity-checking protocol for secure cloud storage. IJIS 14, 1–12 (2014) Yu, Y., Au, M.H., Mu, Y., Tang, S.,
16. Ren, J., Susilo, W., Dong, L.: Enhanced privacy of a remote data integrity-checking protocol for secure cloud storage. IJIS 14, 1–12 (2014)
17. Zhu, Y., Ahn, G.-J., Hu, H., Yau, S.S., An, H.G., Hu, C.-J.: Dynamic audit services for outsourced storages in clouds. IEEE Trans. Serv. Comput. 6(2), 227–238 (2013)
18. Zhu, Y., Wang, H., Hu, Z., Ahn, G.-J., Hu, H., Yau, S.S.: Dynamic audit services for integrity verification of outsourced storages in clouds. In: Proceedings of SAC 2011, pp. 1550–1557 (2011)
19. H. Suo, Z. Liu, J. Wan, et al. “Security and privacy in mobile cloud computing”, Wireless Communications and Mobile Computing Conference, vol. 14, no. 3, pp 655-659, July, 2013.

20. A. F Barsoum, M. Hasan, "Provable Possession and Replication of Data over Cloud Servers", Technical Report, CACR, University of Waterloo, Report 2010/32, 2010.
21. A. F Barsoum, M. Hasan, "On Verifying Dynamic Multiple Data Copies over Cloud Servers", Iacr Cryptology Eprint Archive, vol. 2011, pp 447-448, August, 2011.
22. H. Q Wang, "Proxy Provable Data Possession in Public Clouds", IEEE Transactions on Services Computing, vol. 6, no. 6, pp 551-559, December, 2013.
23. S. Halevi, H. Danny, P. Benny, et al. "Proofs of ownership in remote storage systems", ACM Conference on Computer and Communications Security, pp 491–500, October, 2011
24. M. Gondree, Z. Peterson. "Geolocation of data in the cloud", ACM Conference on Data and Application Security and Privacy, pp 25-36, 2013.
25. J. Zhao, H. X Li, C. Wu, et al. "Dynamic pricing and profit maximization for the cloud with geo-distributed data centers", IEEE INFOCOM, pp 118-126, April, 2014.
26. M. Silic, G. Delac, I. Krka, et al. "Scalable and Accurate Prediction of Availability of Atomic Web Services", IEEE Transactions on Services Computing, vol. 7, no. 2, pp 252-264, June, 2014.
27. J. K Liu, M. H □u, W. Susilo, et al. "Secure sharing and searching for real-time video data in mobile cloud", Network IEEE, vol. 29, no. 2, pp 46-50, April, 2015.
28. C. Gentry, «Toward basing fully hamamarphic encryptian on warst-case hardness,» Advances inCryptology - CRYPTO 2010, val. 6223, pp. 116-137, 2010.
29. C. Gentry, «Fully hamamarphic encryptian using ideal lattices,» Proceedings of the forty-first annual ACM symposium on Theory of computing, pp. 169-178, 2009.
30. C. Gentry, S. Halevi et V. Vaikuntanathan, «A simple BGN-type cryptasystem from LWE,» Advances in Cryptology - EUROCRYPT 2010, val. 6110, pp. 506-522,2010.
31. Rajat Saxena, "Cloud Audit: A Data Integrity Verification Approach for Cloud Computing", Twelfth International Multi-Conference on Information Processing-2016 (IMCIP-2016), Procedia Computer Science 89 (2016) 142 – 151
32. Junyao YE, "Code-based Provable Data Possession Scheme for Integrity Verification in Cloud Storage", 2016 International Conference on Network and Information Systems for Computers
33. Ayad F. Barsoum, "Provable Multicopy Dynamic Data Possession in Cloud Computing Systems", IEEE Transactions On Information Forensics And Security, Vol. 10, NO. 3, MARCH 2015
34. Yasmina BENSITEL, "Secure data storage in the cloud with homomorphic Encryption", IEEE 2016
35. Adil Bouti, "Towards Practical Homomorphic Encryption in Cloud Computing", 2015 IEEE 4th Symposium on Network Cloud Computing and Applications