International Journal of
Intelligent Engineering & Systems

http://www.inass.org/

Reliable and Efficient Distribution of Multicast Session Key for Deduplicated Data in Cloud Computing

Parth Shah[1*] **Amit Ganatra[1]**

[1]*Charotar University of Science and Technology, Changa, India*
* Corresponding author's Email: parthshah.ce@charusat.ac.in

Abstract: Data deduplication is one of the fascinating features of any cloud computing storage service which is generally realized as Cross User Data Deduplication (CUDD). Although it provides optimization which is challenging to achieve due to security concerns. A User always concerns about privacy and confidentiality of the data from honest but curious insiders. Encryption introduces new challenge like key distribution among the group of clients who share the same file and also raises constraints of forward and backward secrecy of the data when any user upload or delete the data. Efficient and secure key distribution along with data integrity verification are the biggest challenges in CUDD. In this work, we have proposed the solution of efficient key management in CUDD along with the data integrity verification. We have provided the solution multicast key distribution using error correcting codes that maintain users' access rights, which is more efficient and reliable.

Keywords: Data deduplication, Rekeying, MDS code; Data integrity, (K, ϴ) uncheatability, Server unforgeability.

1. Introduction

Data deduplication becomes a most important requirement of cloud computing storage applications. It optimizes storages as well as network bandwidth. Deduplication can be categorized as the target-based deduplication handled by the target storage server, while the client remains uninformed of any deduplication that occurs at the server side. This strategy optimizes storage consumption but does not save communication bandwidth. Apart of it, in source-based deduplication, before transmitting data to the server duplication will be checked at the client. Once the duplicates have been found then actual data is not sent. The method improves utilization of storage as well as communication bandwidth.

Providing solution of data deduplication is not that much trivial as it seems. To prevent from unauthorized access clients may encrypt the data using symmetric key encryption algorithms. The algorithm should be efficient in term of execution and should not be dependent on the size of the file. As mentioned earlier, If the data is encrypted then, to provide data deduplication solution is challenging because it complexes the key sharing and the content matching. So key must be shared among the clients having the identical file to provide the confidentiality and data deduplication. Considering this, efficient key distribution algorithm must be used which should take care of key distribution management. Content matching can be solved using some of the Provable Data Possession techniques as given in [1] or some hashing technique.

In cloud computing storage application, deduplication can be implemented as group of users such that clients having identical data will form the multicast group. To provide the confidentiality, group will share session key among all the clients within the group. The key will be initially generated by Key Management Centre (KMC) and first client who have put the data initially. The group memberships change because client may upload new file or delete older file, the KMC releases an independent session key from all the old session keys and retract older key. The rekeying procedure assure the newly joined individuals can't recover the previous sessions, and previous individuals who have left the group can't interact with the present session. Rekeying operation has asymmetric complexity. At the point when new client joins, the KMC can without much of a stretch, multicast the newly created encrypted session key with the current session and unicast it. Thus, computation and communication cost will be very low in case of join. However, the existing session key should not be used, when member leaves to distribute the newly created session key confidentially, since the leaving member knows it. Hence in the case of member leave, the rekeying operation should be focused critically.

Data possession [1-5] is also fundamental concern for clients while using such kind of service. Digital signatures and message authentication codes (MACs), applied to whole file, allow a client in possession of file *F* to verify that it has not been tempered by anyone at server. To verify the integrity of file, most of the methods that exist today use some kind of redundancy. As the integrity verification requires either semantic or syntactic analysis of the information the actual data. Various solutions have been proposed to provide integrity verification of the file. Most of the solutions provides probabilistic solutions which are based on variations of Homomorphic Verifiable Tags (HVT).

In this work, we have proposed efficient and secure deduplication along with efficient multicast key distribution, which can be implemented in remote storage. We have used the solution provided by [1-2] for data integrity verification, [6] for deduplication and [7] for efficient and reliable multicast key distribution. We have provided solution related to data possession for the verification of the integrity of the file, data deduplication and multicast session keys distribution by a central KMC, as those have much less communication complexity which is a very anticipated property in most of the applications [8-13]. The communication complexity is computed by the quantity of bits, which should be communicated between client and server, like hash of the documents for duplication checking, tag values to check the integrity and session keys, though the capacity intricacy is computed by the quantity of bits required by the servers and group members to store hash of the file, tags of the blocks and session key. Another also imperative yet normally under saw, if not disregarded, component is the calculation complexity, which is observed by the number of operations, the server, KMC and group members required to compute and to disseminate and extricate session keys. Hence our solution includes data integrity, data deduplication along with efficient key management in case of user join and leave. Proposed solution focuses on constant verification time and low rekeying overhead.

The paper is organized as follows. Section 2, describes some existing schemes related to proposed work. In section 3, notations are given which are used in proposed solution. Section 4 gives the detailed proposed scheme. Finally, implementation details are given with the performance analysis in section 5 followed by the conclusion.

2. Related Work

An initial solution of secure data deduplication has been proposed by Storer et al [14]. To permit deduplication over common chunks they utilize convergent encryption to perform encryption. In Convergent encryption, the hash of the chunk is used to generate a key. Client encrypting a specified chunk utilizes key generated as a hash value of chunk. So chunk will be encrypted to the same cipher text regardless to who encrypt them. However, there is a substantial disadvantage of utilizing the hash of the key is susceptible to poison attack. Also, targeted collision attacks may be possible due to unverified chunk signatures.

[15] Exhibits information about the contents of files reveals by side channel attack in deduplication. Deduplication can be utilized as a covert channel in which pernicious programming can interact with its control Centre. They have proposed basic components that empower cross-client deduplication while significantly diminishing the danger of information leakage.

The Merkle-tree-based proof of retrievability protocol has been adopted by Halevi et al. [16]. They have used error correcting erasure code which encodes a file and applies the MHT (Merkel Hash Tree) proof over the file. Another solution uses any pairwise independent hash family which is the generic framework. The third solution is an efficient hash family and applies standard Merkel Hash Tree proof over the hash function. It has some drawbacks: (1) the verification accept that document F is tested from a specific kind of distribution (2) the proof uses SHA256 salted as a random function.

Qingji Zheng [1] offers the efficient solution deduplication and demonstrate its security based on the assumption of the Computational Diffie-Hellman (CDH). They analysed and compared the performance of the POSD scheme with existing schemes, which suggest the scheme is as efficient as those schemes. The main advantage of the scheme is that it introduce minor communication overhead.

In this scheme, a client can fully control its key

Table 1. Comparison of various deduplication techniques

Scheme	**[14]**	**[1]**	**[2]**	**[6]**	**[15]**	**[17]**	**Proposed Scheme**
Probabilistic (P)/Deterministic (D)	D	P	P	D	D	D	P
File Level (F)/Block Level (B)	B	F	F	F	F	F	F
Support For PDP/POR	NO	YES	YES	NO	NO	NO	YES
Data Confidentiality	YES	NO	NO	YES	YES	YES	YES
Server Unforgeability	NO	NO	YES	NO	YES	YES	YES
(K, Θ) Cheatability	NO	NO	YES	NO	YES	YES	YES
Rekeying	NO	NO	NO	NO	NO	NO	YES

generation. Also, they have not considered data confidentiality issues.

Youngjoo Shin [2] shows that (k, θ) -uncheatability and server unforgeability are not considered in POSD. They modify the scheme such that the server generates the random value which blended with the keys generated by clients-created. The change is minimized so that their solution maintains the effectiveness while giving more strong security. This scheme doesn't consider the case of encrypted data.

Jia Xu [6] provides improved and comprehensive convergent encryption method (similar to hash-as-a-proof) utilized for security concern. They provide the solution of data confidentiality in bounded leakage model in cloud storage. The scheme has two advantages: (1) it is applicable to any distribution of files, rather than a specific type of distribution; and (2) it uses AES encryption which is considered semantic secure. Additionally, they also aim to protect data privacy against honest-but-curious server.

DupLESS [17] provides an easily-deployed and more secure solution for deduplication with confidentiality. To resists Brute-force attacks and supports Deduplication it uses AES128 cipher and SHA256 algorithms which provide secure outsourced storage. Compared to convergent encryption it gives more security. It has been optimized for low latency which leads to the extra file size but can be reduced as files get larger. Analysis of existing solutions based on the functionality provided is given in Table 1.

3. Preliminaries

3.1 Notations

Let safe primes p and q are k-bit length where $N = p \times q$. Let F_i is composed of n symbols in Z_q. Let the identity f_{id} that distinctively categorizes the file. Let associated with some auxiliary information, denoted by cryptographic information Tag_{int} which is used for auditing data integrity.

3.2 Vandermonde Representation for RS implementation

A widely used class of error control code is Maximum Distance Separable (MDS) code [16]. Let the error control code *(n, k)* and finite field F(g) with g elements [18-19 15-16] having $GF(g)^k \rightarrow GF(g)^n$. Let encoding function $E(d) = c$, where $d = d_1d_2 \ldots d_k$ where $k \leq n$ is the actual message block, and code word block $c = c_1c_2 \ldots c_n$ is, . As per *(n, k)* MDS code there exists a decoding function $D(.)$ such that $D(c_{i1}c_{i2} \ldots c_{ik}, i_1, i_2, \ldots, i_k) = d$ for $1 \leq i_j \leq n$ and $1 \leq j \leq k$, any k symbols of its code word block are used to recover the k actual messages. The process is called erasure decoding.

RS (Reed-Solomon) encoding is an example of MDS. A group of linear equations is used to solve the RS encoding and decoding operation thus it can be used for rekeying purpose. Inverting a coefficient matrix is one of the steps followed by multiplying with it to get the values of the unknowns. If representation has the lower complexity of inverting the coefficient matrix, then the decoding operation will be more efficient. The inversion of Vandermonde matrix is more complex. Therefore, in general, it is considered Vandermonde-matrix based RS codes are less effective [7].

Quite contrary, for RS codes having *(L, 2)* and *(L, 3)*, it is observed that decoding operation is more efficient compared to other representations in Vandermonde representation. The reason being is that the inverse of other matrices for $k = 2$ and $k = 3$ is much complex than the inverse Vandermonde-matrix. The Vandermonde-matrix based RS code is as follows for $k = 3$ [19], [20]:

$$\begin{bmatrix} 1 & i & i^2 \\ 1 & j & j^2 \\ 1 & k & k^2 \end{bmatrix} \begin{bmatrix} d_1 \\ d_2 \\ d_3 \end{bmatrix} = \begin{bmatrix} c_i \\ c_j \\ c_k \end{bmatrix} \qquad (1)$$

Here *i*, *j* and *k* are the identity of members, assigned by KMC at the time of joining the multicast group. To construct the RS codes, finite field $GF(2^m)$ is utilized. So the matrix inverse is represented as

$$\begin{bmatrix} d_1 \\ d_2 \\ d_3 \end{bmatrix} = \begin{bmatrix} \frac{jk}{(i\oplus j)(i\oplus k)} & \frac{ki}{(j\oplus i)(j\oplus k)} & \frac{ij}{(k\oplus i)(k\oplus j)} \\ \frac{j\oplus k}{(i\oplus j)(i\oplus k)} & \frac{k\oplus i}{(j\oplus i)(j\oplus k)} & \frac{i\oplus j}{(k\oplus i)(k\oplus j)} \\ \frac{1}{(i\oplus j)(i\oplus k)} & \frac{1}{(j\oplus i)(j\oplus k)} & \frac{1}{(k\oplus i)(k\oplus j)} \end{bmatrix} \begin{bmatrix} c_i \\ c_j \\ c_k \end{bmatrix} \quad (2)$$

Here d_1 is the multicast session key.

4. Overview of proposed scheme

All the solutions studied in section 2 have not addressed the key management issues for the secure cross user data deduplication. In the proposed solution whenever multiple clients have an identical file which can be deduplicated at the server such that those clients would be considered as a group (or session). Here we are considering the current state of a group as a session. Every time the group memberships change as a result of join (e.g. new client having an identical file) or leave (e.g. client no more want to store that file on the server) of group members, the KMC releases a fresh group key in case of leaving, which is independent of the previous keys. This problem can be considered more precisely as a rekeying problem.

The KMC sends the group key at the initial join, encrypted by a key, shared between the KMC and the group members. So join requires low communication and computation cost. In the case when old member leaves, the current group key cannot be used to do further communication.

We have used erasure decoding of certain Maximum Distance Separable (MDS) code. We follow the basic scheme given in [7]. Here again, a group has n members which will form the multicast group.

Our proposed solution's algorithm is divided into KEY GENERATION and REKEYING, UPLOAD, AUDITINT, DEDUP steps as described below:

Let *p*, *q* be two sufficiently large primes and *G*, G_T be cyclic groups of order *q*. Let $g \in G$ be a generator of *G* and $e: G \rightarrow G_T$ be an admissible bilinear map. Let *F* be a data file consisting of *n* blocks and each block F_i $(1 \le i \le n)$ consist of *m* symbols in Z_q. Let us denote each symbol of F_i as F_{ij} for $1 \le j \le m$. Let *fid* be a unique file id, and let $H_1: \{0, 1\}^* \rightarrow G$ and $H_2: \{0, 1\}^* \rightarrow Z_q$ be hash functions.

KEYGEN: Key generation process involves client and server both for public key and group key generation. While both are involved in a key generation there will not be any problem like server unforgeability and (k, θ)-uncheatability using eq. 4.

We use the construction of a Cross User Data Deduplication scheme given in [1-2]. The first user who uploads file *F*, will create a short secret encryption key k from security parameters (which is obtained from the server) as input.

A pair of public key and private key $\{pk_{int}, sk_{int}\}$ is generated using eq. 3, 4, 5 for integrity verification and group key using eq. 8, 9, 10, 11, 12 for confidentiality. A client and storage server initiates the protocol as follows:

1. The client chooses v_1 and v_2 randomly from Z_p^* such that generated subgroups by v_1 and v_2 are the order of *q*. Choose randomly s_{j1} and s_{j2} from Z_q^* and set
$$z_j = v_1^{-sj1}\, v_2^{-sj2} \bmod p \qquad (3)$$
for $1 \le j \le m$ and send to the server for its contribution.
2. Upon receiving z_j, the server selects σ_{j1} and σ_{j2} uniformly random from Z_q^*, and recomputes the corresponding z_j as follows and send z_j', σ_1 and σ_2 to client:
$$z_j' = z_j\, v_1^{-\sigma}{}_{j1}\, v_2^{-\sigma}{}_{j2} \bmod p \qquad (4)$$
3. The client chooses *u* and *w* uniformly at random from Z_q^* and set
$$z_g = g^w \qquad (5)$$
4. The client initializes $PK_{int} = \{p, q, u, g, v_1, v_2, z_1, \ldots, z_m, z_g, w^{-1}\}$ and the private key $SK_{int} = \{(s_{11}, s_{12}), \ldots, (s_{m1}, s_{m2}), w\}$.
5. Send PK_{int} to storage server and KMC.

To generate the group key, the KMC utilizes *GF(q)* to constructs a non-systematic *(L, n)* MDS code and a secure one-way hash function *H(.)* a secure one-way hash function having codomain of *GF(q)*. The domain of *H(.)* can be an arbitrary to large enough such that *H(.)* satisfies a secure one-way property. The KMC then publically announce both the one-way hash function *H* and the MDS code *C* as eq. 6.

When new member request to the KMC to join it sends a pair (j_i, s_i), to form the multicast group for the first time where j_i and s_i are a positive integer satisfying $j_i \neq j_k$ for all *k*'s, where *k* is a member of the multicast group. Followings are the steps which will be executed between client and KMC.

1. KMC uniformly chooses an element *r* and calculates
$$c_{ji} = h(s_i + r) \qquad (6)$$
$$R = r^{1/w} \bmod q \qquad (7)$$
2. Using all the c_{ji}'s computed in the above step constructs a code word *c* (eq. 1) of the *(L, n)* MDS code *c*, set the j_i symbol of the code word *c* to be c_{ji}.

3. KMC Send a pair (j_i, s_i) to the client and Send $d_2, ..., d_n$ and R to the client.

Above mentioned procedure will be used for rekeying purpose also.

Upon receiving above-mentioned values the client will perform following:

1. Calculates
$$r = R^w \bmod q \tag{8}$$
2. Using a seed key (j_i, s_i) calculates
$$c_{ji} = h(s_i + r) \tag{9}$$
3. Decode the first message symbol d_1 from the $(n-1)$ message symbols $d_2, ..., d_n$ together with its code word symbol c_{ji}.
4. Recover the new session key τ from following
$$\tau = d_1 = c_{ji} \oplus d_2 \oplus ... \oplus d_n \tag{10}$$

UPLOAD: This module runs by a client C and a server S. For pre-processing, the client takes a file F as an input and the secret key sk_{int}, which outputs some auxiliary information Tag_{int} eq. 15-18. This auxiliary information can be used to assess the integrity of F. At the end of the execution, server stores (fid, F, Tag_{int}) received from C. The server may also keep a hash value (step 4) of the F's so as to facilitate the detection of data duplications.

The client will generate a short encoding C_τ using eq. 12 and perform encryption E over F to generate long encoding C_F using eq. 11. The client will send $(hash(F), C_\tau, C_F, hash(C_F), pk_{int})$ to the server, which will keep $hash(F), C_\tau, hash(C_F), pk_{int}$ in small and secure primary storage for the further lookup, and put C_F in the possibly insecure secondary storage. During the tag generation, client will interact with storage server, for the file to be outsourced, to perform the following steps:

1. The encryption algorithm E takes F as an input file and session key τ and outputs C_F using eq. 11, and C_τ as per eq. 12.
$$C_F = E(F, \tau) \tag{11}$$
$$C_\tau = E(\tau, F) \tag{12}$$
2. For each block of data C_{Fij}, where $1 \le i \le n$, the client selects r_{i1}, r_{i2} uniformly at random from Z_q^* and computes using eq. 13-16:
$$x_i = v_1^{r_{i1}} v_2^{r_{i2}} \bmod p \tag{13}$$
$$y_{i1} = r_{i1} + \sum_{j=1}^{m} C_{\mathrm{Fij}} (\mathrm{s}_{\mathrm{j1}} + \sigma_{\mathrm{j1}}) \bmod q \tag{14}$$
$$y_{i2} = r_{i2} + \sum_{j=1}^{m} C_{\mathrm{Fij}} (\mathrm{s}_{\mathrm{j2}} + \sigma_{\mathrm{j2}}) \bmod q \tag{15}$$
$$t_i = (H_1(fid \,||\, i) \bullet u^{H2(xi)})^{\omega} \ (\text{in } G) \tag{16}$$
3. Sends $(f_{id}, C_F, Tag_{int}, C_\tau, hash(F), hash(C_F))$ to the server, where $Tag_{int} = (x_i, y_{i1}, y_{i2}, t_i)$ for $1 \le i \le n$.
4. The server adds an entry $(key = hash(F), value = (hash(C_F), C_\tau))$ to the database.

AUDITINT: This module is executed between server S and auditor, who may be the client or third party auditor. The file fid and the corresponding client's pk_{int} are used as an input. The server's input includes the auxiliary information Tag_{int} associated with F. Basically, this procedure is of challenge-response type, where $chal$ sends by an auditor (step 1-2) and the $resp$ computed by the server eq. 17-20. If $resp$ is valid as per eq. 21-23 an auditor outputs success otherwise fail. Formally, we can write it as

1. The verifier chooses c elements set $I = \{\alpha_1, \alpha_2, ..., \alpha_c\}$ where $\alpha_i \in N$, and coefficient set $\beta = \{\beta_1, \beta_2, ..., \beta_c\}$, where $\beta_i \in Z_q^*$. The verifier sends $chal = (I, \beta)$ as a challenge to the server.
2. Upon receiving $chal$, the server computes
$$\mu_j = \sum_{i \in I} \beta_i C_{Fij} \bmod q \tag{17}$$
for $1 \le j \le m$, and
$$Y_1 = \sum_{i \in I} \beta_i y_{1i} \bmod q \tag{18}$$
$$Y_2 = \sum_{i \in I} \beta_i y_{2i} \bmod q \tag{19}$$
$$T = \prod_{i \in I} t_i^{\beta i} (in\ G) \tag{20}$$
and sends $resp = (\{\mu_j\}_{1 \le j \le m}, \{x_i\}_{i \in I}, Y_1, Y_2, T)$ to the auditor.

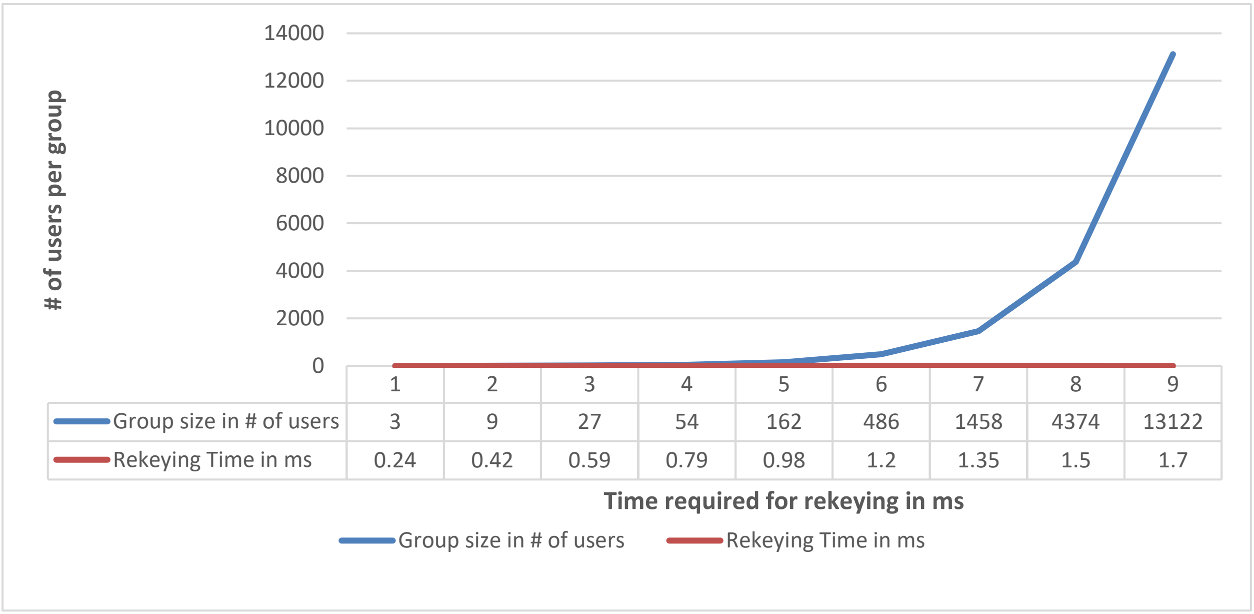
3. The auditor verifies based on received $resp$
$$X = \prod_{i \in I} x_i^{\beta i} \bmod p \tag{21}$$
$$W = \prod_{i \in I} H_1(f_{id} \parallel i)^{\beta i} \bmod p \tag{22}$$
And verifies
$$X = v1^{Y1} v2^{Y2} \prod_{j=1}^{m} z_j^{\mu_j} \bmod p \tag{23}$$

DEDUP: This module is initiated by the client, who upload the identical file F. Client computes and sends h_F to the server (step 1). The server verifies h_F which may be available in its database (step 2). Once server verifies the entry, it send C_F as a response to interact with the client with F as an input. At the end, the client uses τ, generated from the interaction with KMC, it encrypts the F and sends it back (step 3). The server will compare the $hash(C_F)$ provided by the client with the one available in a lookup table. Followings are the steps to perform deduplication.

1. The client will send $h_F = hash(F)$ to the server to check the existence of a file in the storage system.
2. The server looks for the corresponding metadata — $(hash(C_F), C_\tau, PK_{int})$ and send C_F to the client.

Table 2. Time required by the various modules of the system

File Size (KB)	**Tag Calculation (in ms)**		**Response Generation (in ms)**		**Verification (in ms)**	
	Proposed	**Existing [21]**	**Proposed**	**Existing [21]**	**Proposed**	**Existing [21]**
1	5.13	9.29	1.79	5.73	37.19	1.57
11	71.35	120.05	3.99	59.59	35.3	0.94
44	246.61	469.74	8.98	165.4	36.22	1.5
100	419.77	786.74	16.71	321.53	40.11	1.26
425	1719.04	3007.22	62.97	1452.08	33.23	1.98
777	2870.47	5663.69	119.01	2645.33	34.36	2.15
8462	29555.53	48296.02	1301.21	25882.30	32.53	10.5
10113	40898.59	57913.00	1545.28	32304.34	32.82	15.94
51785	191896.97	329224.10	8073.53	163172.62	34.47	117.07
118410	439560.40	727838.39	20632.38	349827.46	39.7	211.18



	1	2	3	4	5	6	7	8	9
Group size in # of users	3	9	27	54	162	486	1458	4374	13122
Rekeying Time in ms	0.24	0.42	0.59	0.79	0.98	1.2	1.35	1.5	1.7

Figure. 1 Time required to regenerate key

3. The client will decode τ and generated C_F. It will calculate *hash(C_F)* and sends it to the storage server for the further verification.
4. The server will mark the entry of client as an owner of the file if *hash(C_F)* matches otherwise decline the access.

5. Implementation and result analysis

We have evaluated the scheme practicality and measured the various parameters like time required to calculate tag, response generation and verification as

Table 2 depict that tag calculation takes exponential increase as file size increases. As this is one-time activity by the client so will not affect furthermore in future. Response generation will be

shown in table 2. All the experiments are implemented using Java Cryptographic Extension (JCE) and tested on Intel i5 processor with 8GB memory running on windows 10 operating system.

As the use of exponential operation is very less, hence the verification process takes the constant amount of time as shown in the *Table 2*. It also depicts that tag calculation and response generation time is quite compared to existing method. This is due to less use of exponentiation operation used in existing system. Though the proposed scheme does not exponentiation operation still provide same level of security.

done at the server side. As it is assumed that server is highly configured so it will not degrade the overall performance. Verification would be done at client side and it is considered that the client is equipped with very less computation power hence time for verification should be less or remain constant. Hence from the table 2, it is observed that verification time remains constant which is independent of the file size.

We have used Advanced Encryption Standard (AES) for the encryption of the file. To use AES for encryption 128 bits key is required hence a finite field of $GF(2^{128})$ is the essential requirement for our solution. Apparently, this field requires 2^{128} elements to have an efficient and meaningful implementation. So it is impossible to have a logarithmic table [16] or an exponential table of this size. Instead, we choose a field of $GF(2^{16})$ to construct RS codes and 8 different iterations. This way, 8 elements in the finite field would be used to compose a 128-bit key. We have used tree-based approach to implement rekeying which is proved to be quite efficient. *Figure 1* shows that time required to rekey based on a different number of users. As per the graph, it is shown that increase in keying or rekeying is linear and will give better performance even the number of users are more.

6. Conclusion and Future work

We have provided the solution of efficient data deduplication along with the data possession. Security weakness of like (k-Θ) cheatability and server unforgeability because of curious server and outside adversaries in the bounded leakage model is overcome by involving server to generate public keys. An MDS codes are utilized to provide the solution of multicast rekeying. The computation complexity of key distribution can be reduced Due to MDS codes this scheme provides balanced and low storage and computation complexity for multicast group key distribution. This work further can be extended using identity-based encryption which requires smaller size public key.

This work can be further enhanced to support block-level data deduplication, which may provide further optimization in terms of storage. Also, data dynamics can be supported to provide more flexibility. Public key distribution can be overcome using Identity-Based encryption.

References

[1] Q. Zheng and S. Xu, "Secure and efficient proof of storage with deduplication", CODASPY'12: *ACM conference on Data and Application Security and Privacy*, pp. 1-12. 2012.

[2] Y. J. Shin, J. Hur, K. Kim, "Security weakness in the Proof of Storage with Deduplication", *IACR Cryptology ePrint Archive*, pp. 1-11. 2012.

[3] Y. Zha, S. Luo, J. Bian and W. Li, "A novel provable data possession scheme based on geographic location attribute," in *China Communications*, vol. 13, no. 9, pp. 139-150, Sept. 2016.

[4] Y. Yu, J. Ni, W. Wu and Y. Wang, "Provable Data Possession Supporting Secure Data Transfer for Cloud Storage," *2015 10th International Conference on Broadband and Wireless Computing, Communication and Applications (BWCCA)*, Krakow, 2015, pp. 38-42.

[5] X. Yu and Q. Wen, "MF-PDP: Multi-function provable data possession scheme in cloud computing," *2014 IEEE 3rd International Conference on Cloud Computing and Intelligence Systems*, Shenzhen, 2014, pp. 597-603. doi: 10.1109/CCIS.2014.7175805

[6] J. Xu, Ee-Chien Chang, and J. Zhou, "Weak leakage-resilient client-side deduplication of encrypted data in cloud storage", In *Proceedings of the 8th ACM SIGSAC symposium on Information, computer and communications security*, pp. 195-206, ACM, May 2013.

[7] L. Xu and C. Huang, "Computation-Efficient Multicast Key Distribution," in *IEEE Transactions on Parallel and Distributed Systems*, vol. 19, no. 5, pp. 577-587, May 2008.

[8] D. R. Stinson, "On Some Methods for Unconditionally Secure Key Distribution and Broadcast Encryption", *Designs, Codes and Cryptography*, vol. 12, pp. 215-243, 1997.

[9] D. R. Stinson and T. van Trung, "Some New Results on Key Distribution Patterns and Broadcast Encryption", *Designs, Codes and Cryptography*, vol. 14, pp. 261-279, 1998.

[10] M. Waldvogel, G. Caronni, D. Sun, N. Weiler, and B. Plattner, "The VersaKey Framework: Versatile Group Key Management," *IEEE J. Selected Areas in Comm.*, vol. 7, no. 8, pp. 1614-1631, Aug. 1999.

[11] S. Mittra, "Iolus: A Framework for Scalable Secure Multicasting", *Proc. ACM SIGCOMM '97*, pp. 277-288, Sept. 1997.

[12] D. M. Wallner, E.J. Harder, and R.C. Agee, "Key Management for Multicast: Issues and Architectures", *IETF Internet draft* https://tools.ietf.org/html/rfc2627, Sept. 1998.

[13] C. K. Wong, M. Gouda, and S.S. Lam, "Secure Group Communications Using Key Graphs," *Proc. ACM SIGCOMM '98*, Sept. 1998.

[14] M. W. Storer, K. Greenan, Darrell D.E. Long, and Ethan L. Miller, "Secure data deduplication", In *Proceedings of the 4th ACM international workshop on Storage*, pp. 1–10, 2008.

[15] D. Harnik, B. Pinkas, A. Shulman-Peleg, "Side channels in cloud services: Deduplication in cloud storage. Security & Privacy", *IEEE* 8(6), pp. 40–47, 2010.

[16] H. S., Harnik D., B. Pinkas, Shulman-Peleg A., "Proofs of ownership in remote storage systems", In *Proceedings of the 18th ACM conference on Computer and communications security*, pp. 491-500, Oct 2011.

[17] M. Bellare, S. Keelveedhi, T. Ristenpart, "DupLESS: Server-Aided Encryption for Deduplicated Storage", In *Proceedings of the 22nd USENIX Security Symposium*, pp. 179-194, USENIX August 2013

[18] F. J. MacWilliams and N. J. A. Sloane, “The theory of error-correcting codes”. I and II. Bull. Amer. Math. Soc. 84, no. 6, pp. 1356—1359, 1978.

[19] J. S. Plank, “A Tutorial on Reed-Solomon Coding for Fault-Tolerance in RAID-Like Systems,” *Software: Practice and Experience*, vol. 27, no. 9, pp. 995-1012, Jan. 1999.

[20] J. S. Plank and L. Xu, "Optimizing Cauchy Reed-Solomon Codes for Fault-Tolerant Network Storage Applications," *Fifth IEEE International Symposium on Network Computing and Applications (NCA'06)*, Cambridge, MA, pp. 173-180, 2006.

[21] G. Ateniese, R. Burns, R. Curtmola, J. Herring, L. Kissner, Z. Peterson, and D. Song, “Provable data possession at untrusted stores”. In Proceedings of the 14th ACM conference on Computer and communications security (CCS '07). ACM, New York, NY, USA, 598-609, 2007.