Turkish Journal of Electrical Engineering & Computer Sciences

http://journals.tubitak.gov.tr/elektrik/

Research Article

Turk J Elec Eng & Comp Sci
() : –

doi:10.3906/elk-2010-133

Zero knowledge based data deduplication using in-line Block Matching protocol for secure cloud storage

Vivekrabinson KANAGAMANI[1,*], **Muneeswaran KARUPPIAH**[2]

[1]Computer Science and Engineering, Mepco Schlenk engineering College, Sivakasi, Tamilnadu

[2]Computer Science and Engineering, Mepco Schlenk engineering College, Sivakasi, Tamilnadu

Received: 30.10.2020 • **Accepted/Published Online:** 20.01.2021 • **Final Version:** ..2021

Abstract: In the area of cloud computing, data deduplication enables the cloud server to store a single copy of data by eliminating redundant files to improve storage and network efficiency. Proof-of-ownership (PoW) is a cryptographic function that verifies the user who really owns the data. Most of the existing schemes have tried to solve the deduplication problem by providing the same encryption key for identical data. However, these schemes suffer from dynamic changes in ownership management. In this paper, we propose an in-line block matching (IBM) protocol based on zero-knowledge proof for deduplication with dynamic ownership management, which eliminates the unauthorized access of sensitive data. In this proposed work, for a new file, the uploader randomly chooses a file encryption key and encrypts the file. The user also computes a unique proof for the uploaded file by dividing the file into number of blocks and stores this proof to the cloud server. The cloud server computes the group key for the ciphertext and re-encrypts it using this group key. The cloud server also does the proof verification against the subsequent uploader for an existing file. The cloud server is honest-but-curious, so the proposed scheme confirms that the cloud server does not know any information about file encryption key even though it plays a proxy role. The result shows that our proposed scheme protects the data from both cloud server and adversaries. Also, the computational cost is comparatively less than other existing schemes.

Key words: Data deduplication, proof-of-ownership, zero-knowledge proof, in-line block matching protocol, dynamic ownership

1. Introduction

In the modern cloud computing era, the enterprises and individuals outsource their data to cloud servers to reduce the burden of deployment and maintenance of local storage devices [1]. To meet the growing demands, the cloud service providers are pushed to perform deduplication [2] on files across the client's data. This increases the storage requirements and the burden on the network data. Deduplication is a data reduction method that helps to eliminate the duplicate copies of data. An indexing scheme is designed to retain only one copy of the data and making provision for the availability of data for all users. This is done by means of maintaining a ownership list (OL) in the form of a table consisting of entries referring to the single copy of data. By this method, any cloud user who tries to upload an existing file will be declined by the cloud server. Instead, it creates an entry in OL, which points to the single copy of the data.

In spite of the advantages of this scheme, there are some security challenges like the consistency of the users' data stored in the server. The cloud server is an honest-but-curious entity, which may gather and inspect

*Correspondence: kvivek@mpcoeng.ac.in

the data stored on the server without the knowledge of the users [3, 4]. To overcome this problem, Douceur [5] proposed a convergent encryption (CE) algorithm at the client side for implementing the deduplication at the sever side. The CE is a symmetric encryption algorithm which uses convergent key for encryption and decryption of the files. The encryption key is obtained by computing the hash value of the file. The identical files produces an identical convergent key and results in producing the same ciphertext which helps the cloud server to perform deduplication. Many existing schemes involved in data deduplication support user revocation and user joining. This process requires a fully trusted third party, which consumes high computing power [6, 7]. In addition, the cloud server cannot differentiate between the authorized/unauthorized users from downloading the data. To overcome the ownership management problem, the concept of proof-of-ownership (PoW) [8] was introduced to verify the users who really owns the file. In our proposed work, we design an in-line block matching (IBM) protocol to solve the duplication problem where the ownership changes dynamically. The in-line deduplication scheme requires less network traffic and storage when compared to the existing methods.

The rest of the paper is organized as follows: Section 2 outlines the related works on proof-of-ownership protocols and different encryption techniques. The cryptographic technique used for the proposed work and the proposed IBM deduplication protocol is illustrated in Section 3. Section 4 analyzes the security features of the proposed model. The performance of the proposed scheme is discussed in Section 5. Section 6 concludes the paper.

2. Related works

The privacy and security concerns of the users' outsourced data increase inevitably as their data is no more stored and controlled by them. Encryption is a simple technique used to protect the users' data, but normal encryption techniques are not suitable for the deduplication process. To solve this issue, Storer et al. [2] and Douceur et al. [5] introduced CE as a cryptographic technique for data deduplication. The convergent encryption is a symmetric encryption algorithm, where the convergent key (CK) for encryption is calculated from the data itself. By using the CE scheme, the encryption produces identical ciphertext for identical data, which makes deduplication easier. However, the CE scheme is exposed to tag consistency problems. To solve the tag consistency problem, the randomized convergent encryption scheme was proposed by Bellare et al. [9]. It is an implementation of message locked encryption that uses an additional tag checking process to verify the integrity of the user data. Shin and Kim [10] proposed an equality predicate encryption scheme. In the scheme, the cloud server possesses the token of each file to know the relation between that file and each other file without the knowledge of the file content. However, this technique is suitable only for single user deduplication.

Wen et al.[11] secured the dynamic update of the data deduplication by proposing a session-key-based convergent key management and a convergent key sharing scheme. The difficulties faced by the scheme is that it is hard to change the session key and the replacement of encrypted convergent key. Hur et al. [6], proposed a secure data deduplication with dynamic ownership management (SDDOM) scheme to solve the dynamic ownership management issue by re-encrypting the ciphertext using the group key. This scheme secures the file from unauthorized users who are not present in data ownership list and the honest-but-curious cloud server with the help of binary key encryption key tree. Jiang et al. [12] proposed a randomized message locked encryption scheme with a randomized tag. For static and dynamic decision trees, the time complexity of the deduplication scheme was reduced with the help of interactive protocols. Yu et al. had worked [13, 16] on data deduplication in cloud storage. To some extent, these schemes tighten the security of the duplicated data, but the complexity of both the system and the convergent key management increases rapidly.

In a client-side deduplication scheme, for the deduplication check, users upload the hash value of a file and halt the upload process if the file has been already stored. Any adversary who knows the hash value of the file could possibly acquire all the data from the cloud. To solve the above-mentioned problem, Halevi et al. [8] proposed the concept of the proof-of-ownership (PoW) model to assure that, only the data owners can retrieve the data from the server. Wang et al. [17] proposed a key-sharing method based on proof of ownership with the help of a Merkle hash tree. For PoW, they divide the file into a set of blocks. However, this scheme suffers from processing large data which generates many leaf nodes, resulting in the problem of communication and computation overhead. For the CK management problem, Li et al. [18] proposed a new scheme called Dekey. In this method users use a ramp secret sharing scheme to divide the CK into multiple shares and distribute them among multiple servers. This scheme suffers from excess communication and computation complexity for retrieving the CK from multiple servers. Scalable and reliable key management using pairing-based cryptography was proposed by Kwon et al. [19]. Here, the CK is divided into three key parts and then distributed is distributed to the external users. This scheme also faces issues when the curious cloud server colludes with an adversary, thereby incurring additional communication and computation overhead.

Merkle-tree based PoW scheme for deduplication was proposed by Xu et al [20]. The limitations of the existing schemes are that it does not verify the ownership of the file upon accessing, and the verification created by a Merkle tree was built using digest rather than the original file. Files in larger size will take long computation time for hash calculation. Xu et al. [21] proposed a cross-user client-side deduplication scheme to protect the sensitive data files from both curious cloud servers and outside adversaries. In this scheme, they use a hash value of the file for proof-of-ownership, which leads the attackers to gain access to entire file by easily guessing the hash value. A formalized private data deduplication scheme to improve the proof-of-ownership was proposed by Ng et al. [22]. In this scheme, the client holds private data and the server holds the summary of the data to verify the ownership of the file. Clients perform the logarithmic operation on every block to build a Merkle tree and verify the user against the server. In our proposed work, we strive to solve the problems mentioned above.

Our contribution: In this paper, we propose an in-line block matching (IBM) scheme based on zero-knowledge for dynamic user management. In our scheme, the users passing the PoW can only obtain the ownership of that file. Our contribution to this paper is summarized as follows:

- First, the initial uploader calculates the file encryption key randomly and computes the key-encryption key with the help of hashing function. The user calculates the unique proof for the file using in-line block matching method and stores it in the database.
- For a subsequent uploader, the server verifies the proof before adding the user information into the ownership list.
- The proposed scheme uses the re-encryption technique to avoid the unnecessary accessing of data from the adversary and curious cloud server.

Compared to other schemes, the proposed work does not require a trusted third party, which reasonably reduces the computational overhead.

3. Proposed work

3.1. Preliminaries

In this section, we describe the cryptographic techniques and the symbols used in our work.

3.1.1. Notations

Table 1 summarizes the notations used in the IBM scheme.

Table 1. List of various symbols and notations in the proposed work.

Sybmol	Meaning
x_i	The secret key of the user u_i
X_i	The public key of the user u_i
F	Original file
E	Key encryption key
K	File encryption key
T	A tag value of the file F
C'_F	The ciphertext of the file F
C_K	The ciphertext of the key K
ID_i	Identity of the user u_i
B_m	The m number of file blocks B
L	Least common multiple (LCM) of two numbers
d_i	Secret key information per user i for the file F
e	Public key information of the file F
G_F	Group key of the file F
P	Unique proof of the file created on initial upload
R, S	Response R and secret value S generated by the subsequent uploader to gain the ownership of files
O	Final output calculated from P and R at server side

3.1.2. Convergent encryption algorithm

Convergent encryption is a commonly used encryption technique for data deduplication [23, 24]. The convergent key (K) is generated by applying the hash function to the file (F), which is denoted by: **CE.GenerateKey**(F) $\rightarrow$ K. The encryption and decryption of the file are denoted as follows:

CE.Encrypt(F, K) $\rightarrow$ C and **CE.Decrypt**(C, K) $\rightarrow$ F, where F and C refer to the original file and ciphertext, respectively. The key is available with the owner of the file. Then, the file gets encrypted using the key and is stored in the cloud, hence no other user can use the file.

3.1.3. Guillou-Quisquater identification protocol

Identification is an interactive process which runs between any two parties, namely a prover (user) and a verifier (server). The Guillou-Quisquater identification protocol is an zero-knowledge protocol [25] which was defined in [26, 27] and designed to provide security against impersonation under passive attacks. Initially, the verifier chooses a random value σ, where σ is the prime exponent of the multiplicative cyclic group $\mathbb{G}$. Let p and q be two large prime numbers and calculate $n = p * q$. Prover wants to prove some value v to verifier; he needs to compute $y = v^{-\sigma}(modn)$. Prover chooses a random value $k \in \mathbb{Z}_q$ and calculates secret as $secret = k^{\sigma}(modn)$. The verifier chooses a random challenge value $ch \in \mathbb{Z}_q$ and sends it to the prover. The prover responds to the challenge by calculating $Response = k.v^{ch}(modn)$. Lastly, the verifier accepts the prover if and only if $secret = Response^{\sigma} * y^{ch}(modn)$.

3.2. Overview of the proposed design

In this paper, we propose a deduplication scheme, which helps to remove duplicate data from the cloud storage where the ownership changes dynamically. The analysis of how the proposed scheme differs from the existing works and the system design of our work are shown in Table 2 and Figure 1, respectively.

Table 2. Comparison of the different deduplication schemes with the proposed scheme.

Scheme	**SDDOM [6]**	**Key-Share [17]**	**Proposed scheme**
Encrypted data deduplication	Yes	Yes	Yes
Access control	Yes	Yes	Yes
User joining	No	Yes	Yes
Proof of ownership	No	Yes, with fixed leaf nodes and time complexity	Yes, with minimized time
Dynamic proof computation	No	No	Yes

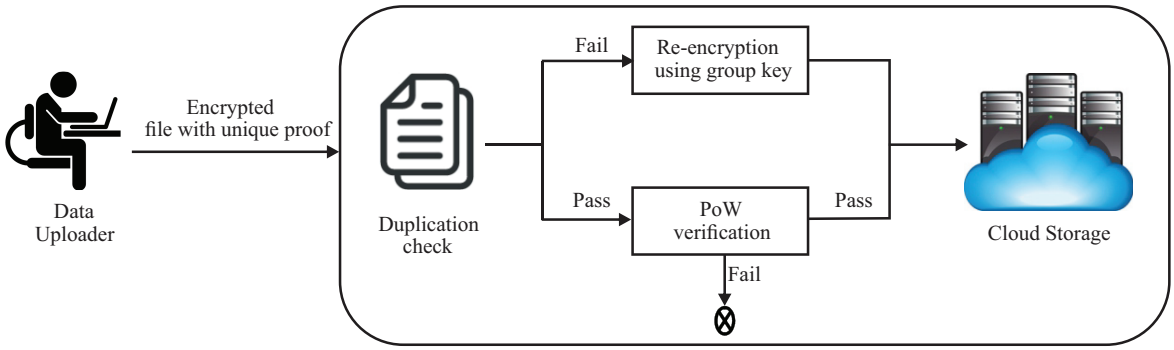

Figure 1. Proposed system design.

The proposed method consists of two entities as shown in Figure 1.

- **Cloud server:** The cloud server is an honest but curious entity, and it provides storage services to users. The service includes elimination of duplicated data and stores only a single copy for each and every file. The cloud server maintains the ownership list for each data stored in the server. The ownership list contains the identities of the user who has permission to access the file.
- **Users:** Cloud users or data uploaders store and retrieve the files whenever needed. Here, the users are classified into initial uploader and subsequent uploader. Any user who uploads the data at first is called the initial uploader, otherwise, they are considered as subsequent uploaders.

3.3. Deduplication using in-line block matching protocol

In the proposed scheme, users are divided into two types as initial uploader (IU) and subsequent uploader (SU).

- **The initial uploader (IU):** With the help of IBM, the initial uploader divides the file into number of blocks and calculates the proofs for those blocks and sends it to the cloud server. The server uses these proofs randomly to verify the subsequent uploading of the same file.

- **The subsequent uploader (SU):** The user who uploads a file which is stored already in the cloud is denoted as a subsequent uploader. In this case, the subsequent uploader needs to verify the ownership of the file that he is trying to upload. To prove the ownership, SU needs to run the IBM protocol along with the server.

The four processes involved in the IBM protocol are explained as follows:

- **UPGen**(F, σ, n) $\rightarrow$ P: The unique proof generation process takes place in client side, which takes as input the file F, the prime number σ, n and generates the proof P. At first, the IU divides the given file into m number of blocks(B) and calculates the proof for each block. It then sends the proof along with the ciphertext of the file, ciphertext of the key, tag of the file, and the ownership-id to the server.
- **ChallengeGen**(r) $\rightarrow$ B, z: Once the uploader is identified as a subsequent uploader, the cloud server randomly choose r number of blocks from the m number of blocks for proof verification process. Next, the server chooses random integer z as a challenge value for each choosen block and sends the values to the subsequent uploader.
- **ResponseGen**(r, B, z) $\rightarrow$ S,R: Using the identification protocol, the subsequent uploader calculates the secret S by choosing a random value s. The SU also calculates the response (R) for the selected blocks using the challenge value z and sends both secret (S) and response (R) values to the server.
- **VerifyProof**(P, S, R) $\rightarrow$ [True, False]: From the inputs, the server verifies the file ownership by computing the product of P, R. Then, the server compares the computed value with S and returns the result. If the result is true, then SU is accepted as a file owner.

3.4. Threat model and security requirements

- **Completeness of the protocol:** The proposed IBM protocol should return true whenever the honest verifier is convinced by an honest prover who posses the original file blocks.
- **Collusion resistance:** Unauthorized cloud users who do not have the valid file blocks should not be able to pass the verification process even if they collude.
- **Zero-knowledge of the protocol:** The proposed protocol should be zero-knowledge, whereby knowing the statement (not secret) is enough for the verifier to visualize that the prover knows the secret.
- **Data privacy:** The cloud server is a honest but curious entity. Therefore, the cloud server should be restricted from accessing the plaintext. Additionally, any unauthorized users (who does not have valid permissions) should also be restricted form obtaining the plaintext.
- **Backward secrecy:** Any cloud user should be restricted from obtaining the plaintext of the outsourced data before uploading the data.
- **Forward secrecy:** Conversely, if a user deletes the data from the cloud, he is no longer the member of the group. The cloud server also needs to restrict these users from obtaining the outsourced data.

3.5. Scheme construction

In this section, we outline the construction of the proposed scheme.

- **Setup:**
  Let $\mathbb{G}$ be a multiplicative cyclic group with prime order t. The algorithm chooses a prime exponent value g, where $g \in \mathbb{Z}_t$. Here, we use the convergent encryption algorithm to perform the first level of encryption. The public and secret keys for each and every user is calculated with the help of ElGamal public-key cryptosystem

[28]. The group key for each and every file is calculated using ElGamal and RSA algorithm. The calculated group keys are secretly distributed to the downloader by using RSA. Let H be a cryptographic hash function used for hashing, where H: $(0, 1)^* \rightarrow \{0, 1\}^n$.

- **User registration:**
Based on successful registration, the server calculates a secret key x and a public key X for the newly joined user. First, the cloud server randomly chooses a number as a secret key x. Secondly, the public key is calculated by X=g^x mod(t).

- **File upload:**
In this proposed work, the file upload process is divided into two types based on the uploader.

(1) **Initial uploader:**
If a user u_1 wants to upload the file F, he should encrypt the file before uploading. At first, the key encryption key (KEK) E is calculated from the file using the hashing algorithm.

$$E = H(F)$$

Secondly, the tag value T for the file is calculated with the help of KEK.

$$T = H(E)$$

Finally, the user chooses a random file encryption key K to encrypt the file F and get C_F. Then, the file encryption key is encrypted with the help of KEK to get C_K.

$$C_F \leftarrow \textbf{CE.Encrypt}(\text{F,K})$$
$$C_K = \text{K} \oplus \text{E}$$

Once the encryption process is done, the user constructs the ciphertext as C= $\{C_F, C_K\}$and uploads the file to server. The format for uploading a file to the cloud server is {C, T, ID_1, P}. Here ID_1 denotes the identity of the owner who owns the file and P denotes the unique proof of the file. The generation of unique proof is explained in Algorithm 1.

Algorithm 1: Unique proof generation by user: **UPGen**(F, σ, n)

Input: The given file F and the prime σ, n
Output: Unique proof P
/* **Unique proof generation at user side** */
F = $\{B_1, B_2, ..., B_m\}$ // Divide the file into m equal blocks
for *i=1 to m* **do**
  $h_i = \text{H}(B_i)$ // calculation of hash value for each block
  $P_i = h_i^{-\sigma} mod(n)$ // calculation of unique proof P
return P = $\{P_1, P_2, ..., P_m\}$

On receiving an upload request from the client, the server initially checks the tag of the current file with the tag of the existing files in the server. If they do not match, the uploader is treated as an initial uploader and the server performs the new file insertion process. Using Algorithm 2, the server calculates the group key G_F for an initial uploader.

Algorithm 2: Group key generation by the server for user u1: **genGF**(e)

Input: The public key e
Output: LCM value L, Secret Key d_1 for user u_1, Group key G_F
/* **Group key generation at server side** */
Choose two random prime numbers a and b for user u_1
$N_1 = a * b$; $\phi(N_1) = (a-1) * (b-1)$ // Calculate N_1 and $\phi(N_1)$
L = lcm(0, $\phi(N_1)$) // Calculation of lcm value L
$\partial = \frac{1}{e} \bmod(L)$ // Calculation of secret key generator value ∂
$d_1 = \partial \bmod(\phi(N_1))$ // Secret key of the file for user u_1
$G_F = d_1 * (X_1 \bmod(t))$ // X_1 is the public key of the user u_1
return L, d_1, G_F

Once the group key has been calculated, the cloud server once again encrypts the ciphertext C_F using the group key G_F and obtains C'_F.

$$C'_F \leftarrow \textbf{CE.Encrypt}(C_F, G_F)$$

Next the server updates the ciphertext as $\{C'_F, C_K$, T, ID, P$\}$ and key information of the file as {e, L, d_1, G_F} and stores into the database. After the successful completion of the upload, the server sends N_1 and d_1 value to the user for group key retrieval.

(2) **Subsequent uploader:**

If user u_2 wants to upload a file which is already stored in the server, we label user u_2 as the subsequent uploader, and the server initiates the proof of ownership verification process. At first cloud server creates a challenge for the subsequent uploader by choosing r number of random blocks, where r $\in$ $1 \leq r \leq m$. The challenge creation is described in Algorithm 3.

Algorithm 3: Challenge generation by server for SU u2: **ChallengeGen**(r)

Input: r number of blocks to be choosen
Output: Block id j, challenge value z
/* **Challenge creation at server side** */
for *i=1 to r* **do**
  $j_i = random(1, 2, ..., m)$ // select a random number from 1,2,...,m
  choose a random integer z_i, where $z \in 1 \leq z \leq 2^n$ // Z is the coefficient for calculating the response
return $j = \{j_1, j_2, ..., j_r\}$, $z = \{z_1, z_2, ..., z_r\}$

Using Algorithm 4, the subsequent uploader computes the response and secret value from the challenge values that are received and sends the results to the server for proof verification.

Algorithm 4: Secret and response creation by SU u2: **ResponseGen**(r, j, z).

Input: No of blocks to be verify r, block id j, challenge value z
Output: Secret S, Response R
/* **Response creation at user side** */
for *i=1 to r* **do**
  choose a random integer s_i, where $s \in 1 \leq s \leq n-1$
  $S_i = {s_i}^{-\sigma} mod(n)$ // Calculation of secret value S
  $h_i = H(B_{j_i})$ // Hash value for each selected block
  $R_i = s_i * {h_i}^{z_i} mod(n)$ // Response calculation
return $S = \{S_1, S_2, ..., S_r\}$, $R = \{R_1, R_2, ..., R_r\}$

Using Algorithm 5, the server calculates the final output O and verifies it against the secret value, then returns the result as either True or False.

Algorithm 5: Proof verification at server side: **VerifyProof**(P, S, R)

Input: Unique proof P, Secret S, Response R
Output: True or False
/* **Proof verification at server side** */
for *i=1 to r* **do**
  $O_i = ({P_i}^{z_i} * {R_i}^{\sigma}) mod(n)$ // Final output
  if $O_i = S_i$ **then**
    $status_i = true$
    return true
  else
    $status_i = false$
    return false

The cloud server generates a new group key from the existing one upon successful verification of the proof using Algorithm 6.

Algorithm 6: Group key generation by server for subsequent uploader: **genGF**$(\{e, L, d, G_F\}, status)$

Input: Group key generation information $\{e, L, d, G_F\}, status$
Output: New group key G'_F, secret key d_2 for u_2, LCM L'
/* **Group key generation at server side** */
if $status_i = true, \forall i = 1, 2, ..., r$ **then**
  Choose two random prime numbers a and b for user u_2
  $N_2 = a * b$; $\phi(N_2) = (a-1) * (b-1)$ // Calculate N_2 and $\phi(N_2)$
  $L' = \text{lcm}(0, \phi(N_2))$ // Calculation of new lcm value L'
  $\partial' = \frac{1}{e} \bmod(L')$ // Calculation of new secret key generator
  $d_2 = \partial' \bmod(\phi(N_2))$ // Secret key of the file F for user u_2
  $G'_F = \frac{G_F * X_2 * d_2}{d} mod(t)$
return L', d_2, G'_F

In Algorithm 6, the generation process takes the key information of the file from the database {e, L, d, G_F} and outputs a new group key G'_F, where e is the public key of the file, L is the LCM value, d is the file secret key of the last subsequent uploader, and G_F is the group key of the file. After the

successful calculation of new group key, the server re-encrypts the ciphertext with new group key G'_F and generates ciphertext C'_F.

$$C_F \leftarrow \textbf{CE.Decrypt}(C'_F,\ G_F)$$
$$C'_F \leftarrow \textbf{CE.Encrypt}(C_F,\ G'_F)$$

Once the encryption process is done, the server updates the ciphertext information as well as adds the identity of the user u_2 to the ownership list. Finally, the server updates the key information of the file F as {e, L', d_2, G'_F}.

- **File download:**

If the user u_1 wants to download the file F from the cloud, the user should send the download request as {F, T, ID_1} to the server. The process of file downloading is described below.

(a) The cloud server initially checks the tag T and identity ID with the files stored in the database. Once the verification succeeds, the cloud server creates the auxiliary information A.

(b) The cloud server calculates η, which is the product of all the N values of the file owners. The value of η is updated whenever the ownership of the file changes.

$$\eta = \prod_{ID \in ownershiplist} N_{ID} \tag{1}$$

(c) Next, the server calculates the auxiliary information A using the formula

$$A = {G_F}^e mod(\eta) \tag{2}$$

where e is a public key of the file and G_F is the group key of the file.

(d) The server sends the ciphertext to the user with the auxiliary information and it is represented as {C'_F, C_K, A}.

(e) At client side, the user calculates the group key G_F from the auxiliary information A using the formula

$$G_F = ((Amod(N_1))^{d_1})mod(N_1) \tag{3}$$

(f) The user calculates the file encryption key K from C_K using $K = C_K \oplus E$, where E is a Key Encryption Key of the file.

(g) Finally, the user decrypts the ciphertext using both G_F, K and gets the original file F.

$$C_F \leftarrow \textbf{CE.Decrypt}(C'_F,\ G_F)$$
$$F \leftarrow \textbf{CE.Decrypt}(C_F,\ K)$$

- **File deletion:**

If the user u_2 wants to delete the file F from the cloud, the server needs to update the ownership information of the file. In the proposed scheme the deletion process is really simple. The cloud server removes the user ownership information from the file by dividing the value of N_2 from η.

$$\eta = \frac{\eta}{N_2} \tag{4}$$

By doing this, even if the attacker retrieves the auxiliary information A, they cannot retrieve the group key G_F and will not be able to decrypt the file.

4. Security analysis

In this section, we discuss the security of the proposed system in terms of threat model which are stated in the Section 3.4.

Theorem 1 *The proposed protocol is complete for honest verifier and prover.*

Proof Assuming that discrete logarithm is hard, the probability of an attacker to retrieve the secret blocks B_i from the unique proof P_i ($P_i = {h_i}^{-\sigma} mod(n)$) is zero, where $h_i = H(B_i)$.

Completeness means that the honest prover (holds the original file blocks) can convince the honest verifier that he owns the file by running the verification protocol. If both the verifier and prover follows the protocol properly, then the final output calculated by the verifier is always equal to the secret value calculated by the prover. This is because

$$\begin{aligned} O_i &= ({P_i}^{z_i} * {R_i}^{\sigma}) mod(n) \\ &= (({h_i}^{-\sigma})^{z_i} * (s_i * {h_i}^{z_i})^{\sigma}) mod(n) \\ &= ({h_i}^{-\sigma z_i} * {s_i}^{\sigma} * {h_i}^{\sigma z_i}) mod(n) \\ &= {s_i}^{\sigma} mod(n) \\ &= S_i \end{aligned}$$

The probability of verifying the user against the server is one. This is because, users who posses original file blocks can only pass the verification process. Thus, our protocol is complete for honest verifier and prover. □

Theorem 2 *The proposed scheme is secure against collusion attack.*

Proof If any unauthorized user with forged file blocks tries to collude with the server to pass the authentication process it results in producing wrong secret value. The probability of server rejecting the adversary proof is 1/2. The possibility of attack is defined as

1. Pick $R_i \in \mathbb{Z}_q$;
2. Guess $z \in \{0,1\}^n$;
3. calculate $S_i = ({P_i}^{z_i} * {R_i}^{\sigma}) mod(n)$;

This shows for a fixed S_i, there will be 2^n distinct R_i values corresponding to 2^n distinct values of z. The probability of guessing each z for each selected block B_i is 2^n. In the proposed protocol, the user needs to interact with the server r times to generate proof. If all r responses are verified, the server admits the user as owner. The probability of the adversary to pass the verification process is 2^{r*n}. This clearly states that, it is computationally intractable for the adversaries to collude with the server to pass the authentication process without having the original file blocks. □

Theorem 3 *The proposed protocol achieves zero-knowledge property against the curious cloud server.*

Proof In an honest-verifier zero-knowledge protocol, if the verifier follows the protocol instruction honestly, then the protocol is perfect. Since the verifier is honest but curious in our system, here we show that the proposed work is a perfect zero-knowledge . An interactive proof not only returns Accept, but it also produces a proof transcript which interleaves the prover's and the verifier's transcript.

To obtain a proof transcript in a perfect zero-knowledge protocol, we do not have to run the protocol between the prover and the verifier, such a proof transcript can be produced via random coin flipping in the polynomial time length of the transcript. An interactive proof is said to be perfect zero-knowledge only when the proof transcript can be produced by a polynomial-time (in the size of the input) algorithm for any common input with the same probability distributions. In the proposed IBM scheme, we use the simulator to produce a simulation of the proof transcript without the interaction of real clients. These proof transcripts are computationally indistinguishable from real proof transcripts produced by interacting with a real client.

For an input P_i, we can construct a polynomial-time simulator $\epsilon Q(P_i)$ as follows:

1. ϵQ initialize transcript as an empty string;

2. For i = 1, 2, . . . , r:
   a. ϵQ picks $R_i \in \mathbb{Z}_q$;
   b. ϵQ picks $z_i \in \{0,1\}^t$; R_i must be uniform in $\mathbb{Z}_q$ for either case of $z_i \in \{0,1\}^t$ and independent of unique proof P_i.
   c. ϵQ computes secret $S_i = (P_i{}^{z_i} * R_i{}^{\sigma}) mod(n)$ Secret S_i should also be uniform and independent of the unique proof P_i;
   d. Transcript $\leftarrow$ Transcript $\|(S_i, z_i, R_i)$ Clearly, Transcript (S_i, z_i, R_i) can be produced by ϵQ in polynomial time, and the elements and its distributions are the same as those in a real proof transcript.

In other words, in a real proof transcript the client can send the data without telling any information about it's private input bits to the server. The elements in the client's transcript are computed based on how the server chooses his random challenge bits. Therefore, the protocol is perfect zero-knowledge even if the server is dishonest. □

Theorem 4 *The proposed scheme assures data privacy against inside and outside adversaries.*

Proof In the proposed work, the cloud server (inside adversary) is honest-but-curious. Here, the possibility for the attacks to be launched can be from a cloud server and an unauthorized cloud user. The form of ciphertext stored in the cloud server is $\{C'_F, C_K, T, \mathrm{ID}, P\}$. The server can easily get the ciphertext C_F by decrypting the ciphertext C'_F. Even though the cloud server has the group key of the files, it is computationally intractable to guess the key encryption key (E) and it cannot acquire the file encryption key K from C_K. The second form of attack can be launched from an unauthorized user (outside adversary) to acquire the plaintext in an unauthorized manner. Even though the unauthorized user acquires the file encryption key K from the data owner, it is computationally infeasible to retrieve the group key and they cannot decrypt the ciphertext C'_F. Thus, our proposed deduplication scheme guarantees data privacy from the cloud server and from an unauthorized cloud user. □

Theorem 5 *The proposed scheme assures the backward secrecy for the stored data.*

Proof When a user tries to store a data that has been stored already in the cloud, he needs to follow the subsequent uploader procedure in order to gain access to the file. Based on the successful verification, the corresponding group key of the file is updated randomly from G_F to G'_F. In addition, the ciphertext $C'_F = Enc_{G_F}(C_F)$ stored in the cloud is re-encrypted with new group key $C'_F = Enc_{G'_F}(C_F)$. Finally, the user is added to the ownership list of the file. Therefore, the cloud users who have the original file blocks can only pass the identity authentication process and obtain the group key. Thus, the backward secrecy of the outsourced data is guaranteed. □

Theorem 6 *The proposed scheme assures the forward secrecy for the stored data.*

Proof If any cloud user deletes or updates an existing data, the server instantly updates the Auxiliary information A and the identity information ID. Initially, the cloud server removes the users' N value from the coefficient η by dividing the N value from η. Therefore, the cloud user will be unable to pass the identity verification process and cannot obtain the group key from auxiliary information A. Thus, our scheme achieves forward security. □

5. Performance analysis

In this section, we compare the performance and efficiency of the proposed scheme with the existing works namely SDDOM [6] and key-share [17]. The testing environment used to compute cryptographic functions are: Intel Core i5 processor, 10GB RAM, Windows10 OS.

Table 3 presents the communication and storage overhead of various schemes. For better security, we set the key and the hashing value to 128-bits. The notations used in Table 3 are defined as follows: S_C is a size of the ciphertext, S_K is a size of the key, S_T denotes the size of the tag information, S_{ID} is a size of ownership list of the file, S_{hF} is a size of the fingerprint of the file, S_P is a size of the unique proof of the file, S_s is the size of the secret random integer, H is a hashing algorithm, Po is the pairing operation, X_G is an XOR operation on group G, S_{kw} is a size of the data owned key for the file F, D_G and M_G are division and multiplication operation on group key G and mod is a modular operation on G.

Table 3. Communication and storage overhead.

Scheme	Communication Overhead			Storage Overhead	
	Upload message size	Download message size	Rekeying Operation	Key Size	Tag Size
SDDOM [6]	S_C+S_K+S_T+S_{ID}	S_C+S_K+S_T	(n-m) log $\frac{n}{n-m}$ S_K	(log n+1)S_K	S_T
Key-Share [17]	S_C+S_K+S_T+S_{ID}+S_{hf}	S_C+S_K+S_s	2H+2Po+X_G	S_K+S_{kw}+S_s	S_T
Proposed Scheme	S_C+S_K+S_T+S_{ID}+S_P	S_C+S_K+S_G	D_G+2M_G+mod	S_G+S_K	S_T

The results in terms of communication and processing time for initial, subsequent upload, and download process are shown in the figures given below.

Figure 2 shows the time taken to upload the file for the first time. In the graph, the x-axis denotes the size of a file and the y-axis denotes the time taken to store the file. The time shown in the graph includes the time taken for unique proof generation and storing the file into the server. From Figure 2, it is understood that the proposed scheme is a promising one.

Figure 3 shows the uploading time for a file which is already stored in the server. In this case, the uploader is treated as a subsequent uploader and he needs to perform proof verification process against the

server. In the graph, the results include the time taken to verify the proof against the server and updating the group key information as well as re-encrypting the stored file in the server. Compared to existing techniques, the proposed scheme takes lesser time for computation of PoW and group key.

The results so far discussed are the analysis of the file uploading process for two different uploaders. The computational analysis of the file downloading process is shown in Figure 4.

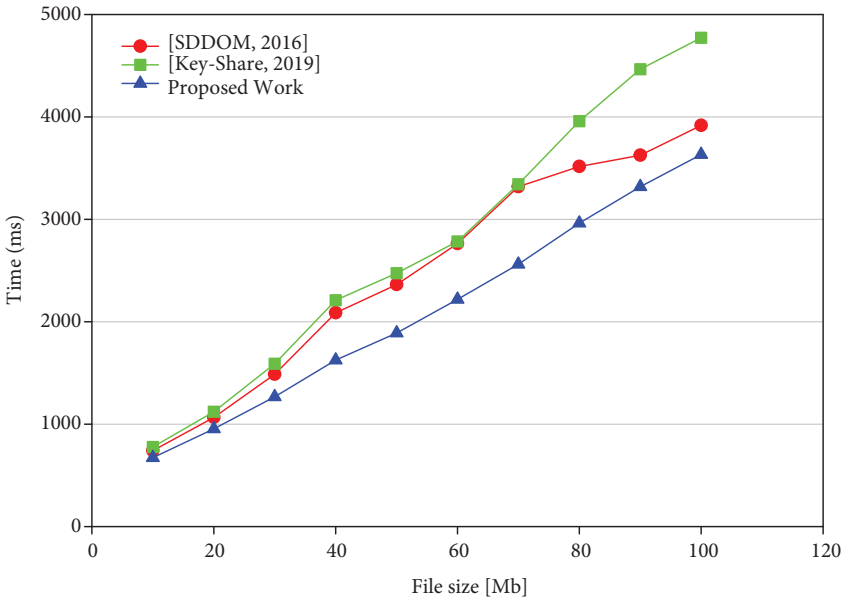

Figure 2. File uploading time for initial uploader

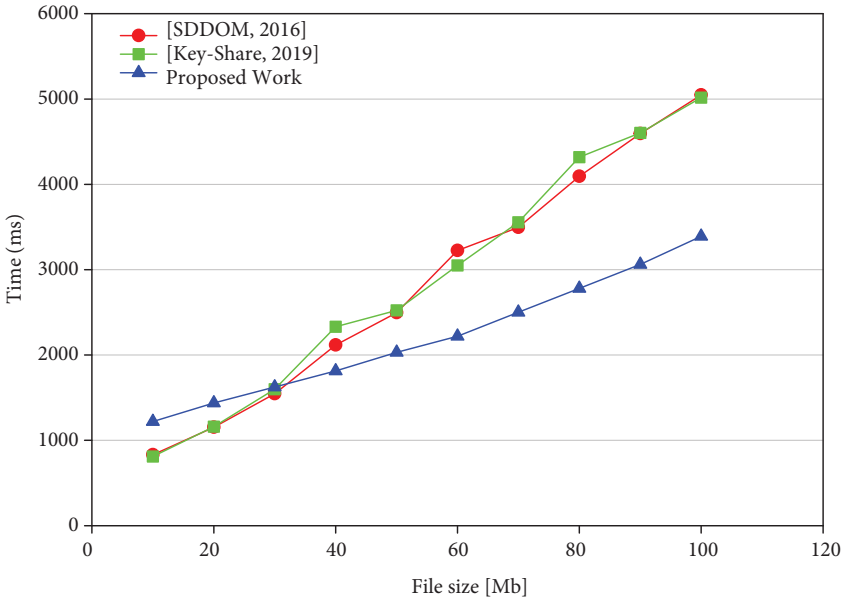

Figure 3. File uploading time for subsequent uploader

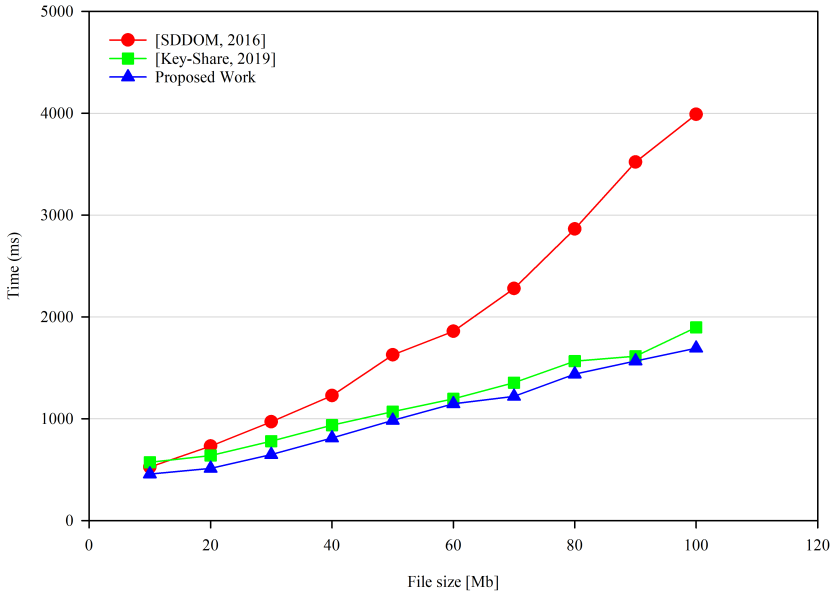

Figure 4. Computation time for file download

Compared with the existing schemes, the proposed scheme has access control mechanisms to verify whether the cloud user is present in the ownership list before downloading. It also uses auxiliary information A to avoid decrypting of file from nondata owners. The results clearly state that the proposed scheme performs better.

6. Conclusion

In this paper, we propose an in-line block matching based data deduplication scheme with dynamic user management. Users encrypt their data using convergent encryption. Using in-line block matching protocol, the server generates unique proof and calculates the group key and re-encrypts the file using the group key. In the subsequent uploader case, the user verifies the proof against the server without leaking any information to the server and re-encrypts using a new group key. Therefore, the confidentiality of the file is assured. We have reduced the communication overhead with the help of the access control techniques, and we also maintain a ownership list to prevent the accessing of ciphertext from unauthorized users. The security analysis shows that the proposed scheme protects the file from being accessed by cloud servers and adversaries. The analysis of the proposed scheme also proves that the probability of detecting the users' misbehavior is high. Our scheme outperforms the existing deduplication schemes in terms of computational time, communication and storage overhead.

References

[1] Liu X, Deng RH, Choo KKR, Weng J. An efficient privacy preserving outsourced calculation toolkit with multiple keys. IEEE Transactions on Information Forensics and Security 2016; 11 (11): 2401-2414. dio: 10.1109/TIFS.2016.2573770

[2] Storer MW, Greenan K, Long DDE. Secure data deduplication. In: Acm International Workshop on Storage Security and Survivability 2008; 1-10. dio: 10.1145/1456469.1456471

[3] He D, Kumar N, Chen J, Lee CC, Chilamkurti N, Yeo SS. Robust anonymous authentication protocol for healthcare applications using wireless medical sensor networks. Multimedia Systems 2015; 21: 49-60. doi: 10.1007/s00530-013-0346-9

[4] Zhang Y, Xu C, Li H, Yang K, Zhou J, Lin X. Healthdep: an efficient and secure deduplication scheme for cloud-assisted ehealth systems. IEEE Transactions on Industrial Informatics 2018; 14 (9): 4101-4112. dio: 10.1109/TII.2018.2832251

[5] Douceur JR, Bolosky WJ, Theimer MM. Encryption systems and methods for identifying and coalescing identical objects encrypted with different keys. US Patent 7266689; 2007

[6] Hur J, Koo D, Shin Y, Kang K. Secure data deduplication with dynamic ownership management in cloud storage. IEEE Transactions on Knowledge and Data Engineering 2016; 28 (11): 3113-3125. doi: 10.1109/TKDE.2016.2580139

[7] Yan Z, Ding W, Yu X, Zhu H, Deng RH. Deduplication on encrypted big data in cloud. IEEE Transactions on Big Data 2016; 2 (2): 138-150. doi: 10.1109/TBDATA.2016.2587659

[8] Halevi S, Harnik D, Pinkas B, Peleg AS. Proofs of ownership in remote storage systems. in: Proceedings of the 18th ACM Conference on Computer and Communications Security; Chicago, Illinois, USA; 2011. pp.491-500

[9] Bellare M, Keelveedhi S, Ristenpart T. Message-locked encryption and secure deduplication. In: Proceedings of International Conference on the Theory and Applications of Cryptographic Techniques; Athens; 2013. pp.296-312

[10] Shin Y, Kim K. Equality predicate encryption for secure data deduplication. In: Proceedings of Conference on Information Security and Cryptology; Seoul, Korea; 2012. pp.64-70

[11] Wen M, Ota K, Li H, Lei J, Gu C et al. Secure data deduplication with reliable key management for dynamic updates in CPSS. IEEE Transactions on Computational Social Systems 2015; 2 (4): 137-147. doi: 10.1109/TCSS.2015.2514088

[12] Jiang T, Chen X, Wu Q, Ma J, Susilo W et al. Secure and efficient cloud data deduplication with randomized tag. IEEE Transactions on Information Forensics Security 2017; 12 (3): 532-543. doi: 10.1109/TIFS.2016.2622013

[13] Li Y, Yu Y, Min G, Ni J, Susilo W. Fuzzy identity-based data integrity auditing for reliable cloud storage systems. IEEE Transactions on Dependable Secure Computing 2019; 16 (1): 72-83. doi: 10.1109/TDSC.2017.2662216

[14] Xue L, Yu Y, Li Y, Au MH, Du X et al. Efficient attribute-based encryption with attribute revocation for assured data deletion. Information Sciences 2019; 479: 640-650. doi: 10.1016/j.ins.2018.02.015

[15] Yu Y, Li Y, Yang B, Susilo W, Yang G et al. Attribute-based cloud data integrity auditing for secure outsourced storage. IEEE Transactions on Emerging Topics in Computing 2017; 8 (2): 377-390. doi: 10.1109/TETC.2017.2759329

[16] Yu Y, Xue L, Li Y, Du X, Guizani M et al. Assured data deletion with fine-grained access control for fog-based industrial applications. IEEE Transactions on Industrial Informatics 2018; 14 (10): 4538-4547. doi: 10.1109/TII.2018.2841047

[17] Liang W, Baocang W, Wei S, Zhili Z. A key-sharing based secure deduplication scheme in cloud storage. Information Sciences 2019; 504: 48-60. doi: 10.1016/j.ins.2019.07.058

[18] Li J, Chen X, Li M, Li J, Lee PPC et al. Secure deduplication with efficient and reliable convergent key management. IEEE Transactions on Parallel and Distriuted System 2013; 25 (6): 1615-1625. doi: 10.1109/TPDS.2013.284

[19] Kwon H, Hahn C, Koo D, Hur J. Scalable and reliable key management for secure deduplication in cloud storage. in: 2017 IEEE 10th International Conference on Cloud Computing; Honolulu, CA, USA; 2017. pp.391-398.

[20] Xu J, Chang EC, Zhou J. Leakage-resilient client-side deduplication of encrypted data in cloud storage. Cryptology ePrint Archive, Report 2011/538; 2011. http://eprint.iacr.org/.

[21] Xu J, Chang EC, Zhou J. Weak leakage-resilient client-side deduplication of encrypted data in cloud storage. ASIA CCS '13: Proceedings of the 8th ACM SIGSAC symposium on Information, computer and communications security; Hangzhou, China; 2013. pp.195-206. doi: 10.1145/2484313.2484340

[22] Ng WE, Wen Y, Zhu H. Private data deduplication protocols in cloud storage. in: Proceedings of the ACM Symposium on Applied Computing; SAC 2012, Riva, Trento, Italy; 2012. pp.441-446.

[23] Douceur JR, Adya A, Bolosky WJ, Simon P, Theimer M. Reclaiming space from duplicate files in a serverless distributed file system. In: Proceedings 22nd International Conference on Distributed Computing Systems; Redmond, WA; 2002. pp.617-624.

[24] Li J, Li YK, Chen X, Lee P, Lou W. A hybrid cloud approach for secure authorized deduplication. IEEE Transactions on Parallel and Distributed Systems 2015; 26 (5): 1206-1216. doi: 10.1109/TPDS.2014.2318320

[25] Yang C, Zhang M, Jiang Q, Zhang J, Li D etal. Zero knowledge based client side deduplication for encrypted files of secure cloud storage in smart cities. Pervasive and Mobile Computing 2017; 41: 243-258. doi: 10.1016/j.pmcj.2017.03.014

[26] Wenbo M. Modern Cryptography: Theory and Practice. Prentice Hall; 2003

[27] Guillou L, Quisquater JJ. A paradoxical identity-based signature scheme resulting from zero-knowledge. Advances in Cryptology – CRYPTO'88; 1998. pp.216-231. doi: 10.1007/0-387-34799-2

[28] Elgamal T. A public key cryptosystem and a signature scheme based on discrete logarithms. IEEE Transactions on Information Theory 1985; 31 (4): 469-472. doi: 10.1109/TIT.1985.1057074