Secure Batch Deduplication Without Dual Servers in Backup System

Haoyu Zheng, Shengke Zeng, Hongwei Li, *Member, IEEE*, and Zhijun Li

Abstract—Cloud storage provides highly available and low cost resources to users. However, as massive amounts of outsourced data grow rapidly, an effective data deduplication scheme is necessary. This is a hot and challenging field, in which there are quite a few researches. However, most of previous works require dual-server fashion to be against brute-force attacks and do not support batch checking. It is not practicable for the massive data stored in the cloud. In this paper, we present a secure batch deduplication scheme for backup system. Besides, our scheme resists the brute-force attacks without the aid of other servers. The core idea of the batch deduplication is to separate users into different groups by using short hashes. Within each group, we leverage group key agreement and symmetric encryption to achieve secure batch checking and semantically secure storage. We also extensively evaluate its performance and overhead based on different datasets. We show that our scheme saves the data storage by up to 89.84%. These results show that our scheme is efficient and scalable for cloud backup system and can also ensure data confidentiality.

Index Terms—Batch Deduplication, Cloud Storage, Semantic Security, Brute-force Attack

1 Introduction

Cloud storage provides the customers and enterprises big space to outsource their data. With data volume proliferating quickly [1], however, it puts a heavy strain on cloud service providers. Redundancy is the main problem faced by cloud service providers as they store large amounts of duplicate data [2] [3]. Data deduplication technique is a solution to reduce the storage cost. Therefore, it has attracted more and more attention from academia and industry, i.e., Google Drive [4], Dropbox [5], and Memopal have employed the deduplication technique to reduce the storage overheads [6] and improve the performance on their backup systems [7].

However, for the sensitive data stored in the cloud, the deduplication is not trivial. In case of curious cloud service providers, their data should be encrypted to avoid the leakage of users' privacy [8] [9] [12] [13] [14] [15] [16]. Obviously, different users produce different ciphertexts although they have the same plaintexts. It makes cross-users deduplication impossible. Furthermore, after achieving cross-user deduplication, one more problem is to retrieve the ciphertext encrypted by other keys. Therefore, encrypted deduplication was proposed. Nowadays, encrypted data deduplication schemes can be broadly classified into serverless deduplication and server-aided deduplication. *Convergent Encryption* (CE) [10] is the first encrypted deduplication scheme that has been proposed without the help of additional servers in which the hash value of the raw data is the encryption key. In this way, the same data results the identical ciphertext. Although CE [10] realizes the cross-users deduplication, however, it is vulnerable to offline brute-force attacks [8] and tag-consistency problem [11] for its deterministic output. Then *Message-Locked-Encryption* (MLE) [11] was proposed and each chunk of data is encrypted by the key which derived from that chunk. However, it also suffers from the offline brute-force attack and information leakage [9] for its deterministic tag. Therefore, Bellare et al. [17] proposed the first server-aided secure deduplication named DupLESS to be against the offline brute-force attacks. However, the requirement of the aid of additional servers is a strong assumption. It is vulnerable to the single-point-of-failure. Moreover, current server-aided schemes [18] [19] require fully trusted entities. If entities are compromised, the security can not be guaranteed. Therefore, how to achieve secure (i.e., against brute-force attacks) encrypted deduplication without the help of additional servers is a challenge.

Hence, Liu et al. [20] proposed the first deduplication scheme without additional servers to resist against brute-force attacks. Their scheme makes use of the high collision rate of the short hash to be resistant to the offline brute-force attack. An underlying *Password Authenticated Key Exchange* (PAKE) [21] protocol is run by the clients to produce the sharing encryption key. The interactive steps are necessary as it does not require the extra servers. However, we note that it can only check the duplication one by one. It cannot be extended to support batch deduplication of massive data. For the backup system with the extremely large amounts of data, their scheme is not practical. Ma et al. [36] proposed a lazy deduplication for disk storage in local machines. It buffered the fingerprints of chunks that are used to execute on-disk lookups in batches. It can highly reduce the disk I/O. Obviously, it does not support encrypted deduplication for cloud storage. Apparently, how to perform efficient duplicate checking in cloud with the extremely large amounts of encrypted data is also a challenge. We notice that most

H. Zheng is with Xihua College, Xihua University, Chengdu, 610039, China.
S. Zeng is with the school of Computer and Software Engineering, Xihua University, Chengdu, 610039, China. She is also with the School of Computer Science and Engineering, University of Electronic Science and Technology of China, Chengdu 611731, China.
H. Li is with the School of Computer Science and Engineering, University of Electronic Science and Technology of China, Chengdu 611731, China.
Z. Li is with the MIG Group, Cisco Systems Canada Co., Ottawa, ON, K2K 3E8, Canada.
**Corresponding Author: Shengke Zeng; Email: zengsk@mail.xhu.edu.cn.*
*Manuscript received ****, ****; revised ****, ****.*

deduplication schemes can only check the duplication files one by one. It's very time-consuming and inefficient for the cloud system to scan all the data set which incurs a huge delay. Therefore, implementing an encrypted deduplication which supporting batch checking is significant.

In the era of big data, we need to store and manage enormous amounts of private data in cloud server. Therefore, improving the storage performance and security are necessary. If the server can only compare 2 files in a session for deduplication, it is very inefficient obviously. How to support batch checking on duplication for the massive amounts of storage data and achieve semantically secure encryption without multi-server collaboration is our goal to deduplicate efficiently and securely in the modern backup system.

1.1 Contribution

We focus on the deduplication efficiency in backup system with mass data and present a batch deduplication with encrypted files in this paper. Our scheme supports semantic security without requiring the aid of additional independent servers hence it is more practical. Our main contributions can be summarized as follows.

Firstly, our scheme achieves semantic security and prevents online/offline brute-force attacks without dual-server collaboration. We use a short hash followed by the idea of [20] and a group key agreement protocol as our building block. The short hash has 2 sides of purpose. The one is to avoid brute-force guess attack for its high collision rate. Another one is to filtrate the potential identical files owners who will be invited by the cloud server provider to determine the real identical files. The group key agreement is used to randomize the group tag against the outsider (i.e., the corrupted server).

Secondly, our scheme provides the batch checking for the underlying identical files. We use a symmetrical encryption to separate the identical files. The encryption key is the hash value of the unloading file. Each client chooses a random value to be encrypted under this key and forwards to others by the cloud server. Then each client decrypts and gets the random values. Finally, the group tag is a hash value of these obtained values. Obviously, the server provider separates the identical files by these tags without checking them one by one.

Thirdly, we design a *map* and *set* programming model and leverage it to perform server-side deduplication. Our experiments not only simulate our scheme to deduplicate on three different datasets, but also result to the optimal threshold selection to perform our solution effectively. We also analyze the performance and overhead of file operations. These results show that the proposed scheme is secure and efficient.

1.2 Organization

This paper is organized as follows. In Section 1, we give introduction and problem statement. In Section 2, we present the preliminaries of this work. In Section 3, we present the system model and attack model of our scheme. In Section 4, we present our concrete construction in detail with further discussions and the security analysis is provided in Section 5. In Section 6, we show the performance evaluations and present the experimental results. Finally, we conclude the paper in Section 7.

2 PRELIMINARIES

2.1 Deduplication

Deduplication is a compression technique that can effectively eliminate redundant data. It just saves one copy of duplicate files or chunks. By using some pointers, other duplicate data is referred to the only one copy. Based on the data units processed, deduplication can be divided into *file-level* deduplication and *chunk-level* deduplication. Depending on where the deduplication takes place, it can be categorized into *client-side* deduplication and *server-side* deduplication. Our work focuses on *file-level* and *server-side* deduplication, although it can be extended to support client-side deduplication.

2.2 Encrypted Deduplication

Encrypted data deduplication preserves the efficiency of deduplication while protecting the privacy and security of users. Douceur et al. [10] proposed the notion of *Convergent Encryption* (CE) to first handle the deduplication with data confidentiality. As explained above, CE and its varaint [29] [30] [31] have been realized and deployed in many systems. But they are vulnerable to offline brute-force attack [8] for the predictable files, as the adversary can brutely force to compute the hash value of all possible plaintexts to check the underlying plaintext. Therefore, its security is only achieved when the target data comes from an unpredictable space or that space is large enough.

Bellare et al. [17] proposed DupLESS, which realizes the server-aided MLE [12] to overcome the offline brute-force attack. However, its security is at the cost of managing a semi-trusted key server. The user runs an interactive protocol with the server. The server not only generates the convergent key, but also uses a *global secret* as an input when computing convergent key. By doing so, the convergent key is independent of data itself. As long as the global secret is inaccessible to attackers, high security can be ensured. However, once the global secret is compromised, DupLESS is reduced to the conventional MLE. In addition, to strengthen the confidentiality, DupLESS encrypts data with convergent keys obtained from a key-server via *oblivious pseudo-random function* (OPRF) [22]. Such that, users do not leak any information about their data to the server. However, DupLESS induces considerable computational overheads in deduplication as OPRF protocol is time-consuming. Several works [19] [32] [33] realize secure deduplications depending on the idea that requires the aid of additional independent servers. However, server-aided schemes ensure the security under the strong assumption which is not practical. To handle this problem, Liu et al. [20] proposed the first single-server deduplication scheme to achieve semantical security. It adopts the short hash value which has high collision rate to avoid the guess attack. However, it requires 6 rounds to communicate during one upload which is impractical to implement in applications. Li et al. [23] proposed the *Tunable Encrypted Deduplication* (TED) scheme, which allows users

to balance the trade-off between storage efficiency and data confidentiality. TED is based on the MLE, but the derivation key not only from the chunk itself but also from the number of identical chunk copies, which makes TED against frequency analysis attack [34]. Zhang et al. [24] proposed a secure password-protected MLE key scheme called SPADE, which can resist compromised key servers by supporting periodical changes of the key servers and free users from the key management problems. By using re-encryption, several schemes are also proposed. For example, Yuan [25] proposed a secure data deduplication scheme by using re-encryption based on the *convergent all-or-nothing transform* (CAONT) [28] and randomly sampled bits from the bloom filter which enhances the security of REED [26]. Attribute-based encryption is also applied to encrypted deduplication. Cui et al. [27] proposed an attribute-based deduplication in hybrid cloud setting, which achieves the semantic security for data confidentiality but with additional servers. They also exploited zero-knowledge proof of knowledge and commitment scheme for solving data consistency problems [11]. Zhang et al. [38] proposed the *Secure and Efficient Data Deduplication* (SED) scheme, which uses the collaboration between joint clouds to complete the deduplication. It does not require a trusted key server, but in fact still requires the collaboration with other servers. We summarize some works in the following table 1.

2.3 Hash Collisions

Hash functions are simply functions that take inputs of some length and compress them into fixed-length outputs. A hash function H takes a string $x \in \{0,1\}^*$ as input and outputs a string $H(x) \in \{0,1\}^n$ where n is the length. We assume that hash function with long outputs is collision resistant but short hash function has many collisions. We make use of the short hash function to avoid guess attack and improve efficiency.

2.4 Group Key Agreement Protocol

Key agreement protocol is used to calculate a shared key between participants without being known to outsiders. Original key agreement protocol is performed between 2 parties and it can be extended to a group. BD protocol was proposed by Burmester and Desmedt [35] to realize a constant-round group key agreement. The members in a group can obtain a shared group key while others know nothing. Suppose there are n members want to share a group key. These members form a ring to compute the common group key through BD protocol. The parameters are: g is a generator of a cyclic group G with prime order p, for each member M_i, $i = 1, \ldots, n$:

- input a random value $t_i \leftarrow Z_p$, compute and broadcast $z_i = g^{t_i} \mod p$.
- compute and broadcast $X_i = (z_{i+1}/z_{i-1})^{t_i} \mod p$.
- compute the group shared key $K = z_{i-1}^{nt_i} \cdot X_i^{n-1} \cdot X_{i+1}^{n-2} \ldots X_{i-2} = g^{t_1t_2+t_2t_3+\cdots+t_nt_1} \mod p$

It is easy to see that the interaction step in BD protocol is constant no matter what number of members are in this group and it is secure under Diffie-Hellman assumption.

3 Secure Deduplicated Backup System Model

3.1 Design Goals

1) *How to achieve secure encrypted deduplication without the help of additional servers on the server side*? Current server-aided schemes require fully trusted entities. If entities are compromised, the security can not be guaranteed. Therefore, a semantically secure deduplication scheme against brute-force attacks without strong assumptions is necessary.

2) *How to perform efficient duplicate checking in cloud with the extremely large amounts of data*? We notice that most deduplication schemes can only check the files one by one. It's very time-consuming and inefficient for the cloud system to scan all the data set which incurs a huge delay. Therefore, implementing an encrypted deduplication which supporting batch checking is significant.

3) *How to support the client to retrieve the encrypted files which are deleted for the duplication*? Since these files are encrypted under the different keys, when the client requires to download files, the secure cross user key transfer is necessary.

3.2 Backup System Model

The backup system consists of central storage service provider and a set of local storage servers in general. The local storage servers run as the clients to duplicate their files to the central storage service provider for the redundant storage. Once the data disaster occurs, the local server retrieves the files from the central server. Undoubtedly, the storage load of the central storage service provider is heavy and it is urgent to delete the identical files uploaded by different local storage servers. On the other hand, these data provided by local servers are fused in the central server, the encryption is necessary for the data privacy. Therefore, the data is encrypted stored in the servers. The backup system model is as shown in Fig.1. From a realistic viewpoint, these local servers do not communicate with each other directly. Instead, they communicate with the central server which forwards the messages to realize the communication of local servers. We assume the communication channels are secure, and attackers can not eavesdrop the channels.

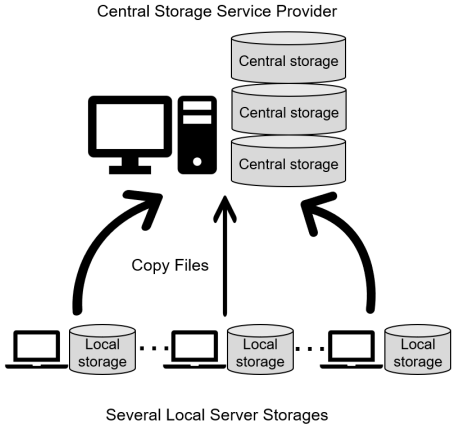

Fig. 1. Backup System Model

TABLE 1
Comparison of Deduplication Schemes

Schemes	brute-force attack resistance	check model
[10]	✖	*one by one*
[17]	*server-aided based*	*one by one*
[38]	*blind signature based*	*one by one*
[39]	*blind signature based*	*one by one*
[18]	*server-aided based*	*one by one*
[19]	*server-aided based*	*one by one*
[20]	*without dual-server*	*one by one*
Our Goals	**without dual-server**	**batch checking**

3.3 Attack Model

For the data privacy, we focus on the attack model in the backup system for the server-side. In server-side deduplication, all files are uploaded and stored in the server, which then deletes the duplicates. The client $\mathcal{C}$ (i.e., the local server in the backup system) is unaware of deduplication. Therefore, the compromised $\mathcal{C}$ is not considered in this threat model. In our model, we pay attention to the malicious central server $\mathcal{S}$, which stores the backup files of the local servers. $\mathcal{S}$ may guess the files stored in it though they are encrypted. Indeed, if these files are predicable, the deterministic encryption results help $\mathcal{S}$ to determine whether a certain file is stored or not. In general, there are two kinds of threats by $\mathcal{S}$.

- *Online brute-force attack*: The compromised *active* $\mathcal{S}$ can masquerade as the local server, say the client $\mathcal{C}$ to run the interactive deduplication protocol to check the underlying possible files by observing which one causes deduplication.
- *Offline brute-force attack*: The compromised *passive* $\mathcal{S}$ can guess its stored files offline from the deterministic tags of the predictable files, even though they are encrypted.

3.4 Ideal Goals

To achieve secure and practical deduplication in cloud server, our scheme should have following features.

- **Security.** The scheme should be resistant to the brute-force attack. Indeed, the encryption files should achieve semantical security and any adversary cannot get the client file content by online/offline brute-force attacks.
- **Efficiency.** Reducing computational and communication overhead is required. Besides, for the huge amounts of data, the duplication check for files one by one is not practicable obviously. Therefore, the batch check is deserved thereby the parallel computation for the underlying files should be supported.
- **Deduplication Effectiveness.** The deduplication effectiveness should be maximized which will detect all duplicates. It is expressed by the deduplication ratio, which requires that the ratio of the number of bytes input and the number of bytes output is maximized.

4 THE PROPOSED SCHEME

In this section, we describe a concrete construction of secure batch deduplication. The files in our scheme are not checked one by one, instead, the central server $\mathcal{S}$ can divide the massive encrypted files into several groups (i.e., in which the files are the same) in *constant* executions. Since the files in a group are the same, $\mathcal{S}$ deletes the copies.

4.1 Overview

In traditional server-side deduplication, when a client wants to upload a file to server, he first sends a hash value h of the file. Therefore the server can check whether the file is in its storage or not. However this approach is insecure, dishonest server can easily launch offline brute-force attacks on the hash value h if the file is predictable.

In our scheme, the clients are the local servers $\mathcal{C}_i$ which make a backup of the stored files to a remote central server $\mathcal{S}$. Our strategy is that, $\mathcal{C}_i$ uploads a short hash value of the file $\mathtt{F}$, i.e., 10 or 20 bits long, to $\mathcal{S}$. For $\mathcal{S}$, it identifies the underlying identical files by the same short hash value. Let $\mathcal{C}_i$s with the same short hash value negotiate a group tag, which is produced by the file $\mathtt{F}$. Obviously, the same file outputs the same tag. In this way, $\mathcal{S}$ can determine the ciphertexts encrypting the same file to a logic group by a constant computation complexity. If the data disaster occurs, $\mathcal{C}_i$ wants to recover the file from $\mathcal{S}$ after deduplication, $\mathcal{S}$ runs a ciphertext transfer protocol with $\mathcal{C}_i$ to help its clients to retrieve the files.

Therefore, our scheme for the secure backup with batch deduplication is divided to 3 phases: *uploading*, *batch duplication check* and *file retrieval*.

4.2 Construction

Our detailed constructions of scheme are described as follows:

- **Uploading**.
  $\mathcal{C}_i$ uploads an encrypted file $\mathtt{C_i} = \mathtt{SE}(\kappa_\mathtt{i}, \mathtt{F})$ with the corresponding short hash value $\mathtt{sh} = \mathtt{SH}(\mathtt{F})$ to $\mathcal{S}$. Note that, $\mathtt{SE}$ is a secure symmetric encryption scheme, $\kappa_\mathtt{i} = \mathtt{H}(\mathtt{k_i}, \mathtt{F})$ is the encryption key generated by the long-term secret key $\mathtt{k_i}$ of $\mathcal{C}_i$ and $\mathtt{H}$ is a cryptographic hash function with onewayness and collision-resistance. $\mathcal{S}$ records them and maintains a tuple of the form $(\mathtt{ID_i}, \mathtt{C_i}, \mathtt{sh})$ for $\mathcal{C}_i$, where $\mathtt{ID_i}$ is the identity of $\mathcal{C}_i$.
- **Batch Duplication Check**.
  When $\mathcal{S}$ wants to check duplication, it first checks the list $(\mathtt{ID_i}, \mathtt{C_i}, \mathtt{sh})$ for the same $\mathtt{sh}$ and updates the list as $(\{\mathtt{ID_i}, \mathtt{C_i}\}_\mathtt{n}, \mathtt{sh})$, where $\mathtt{n}$ is the client number who has the same $\mathtt{sh}$. Then $\mathcal{S}$ communicates with $\mathcal{C}_i$ in the list $(\{\mathtt{ID_i}, \mathtt{C_i}\}_\mathtt{n}, \mathtt{sh})$ to negotiate a group tag. That

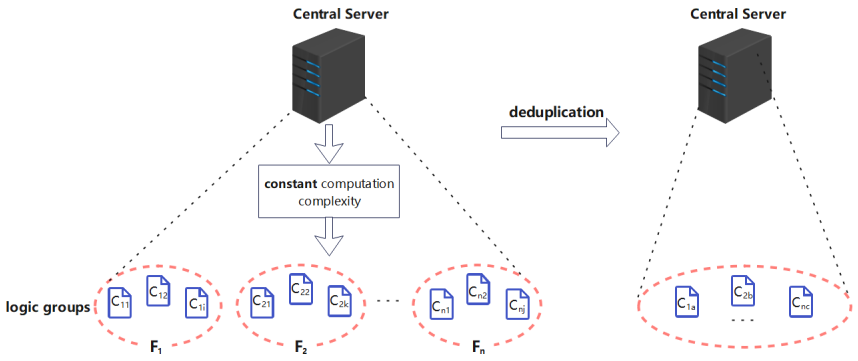

Fig. 2. Batch Deduplication

is, the n clients which have the same sh to determine whether the encrypted files in C_i are identical or not. If yes, they form a logic group to share one copy of this file as shown in Fig.2. This logic group is formed as follows:

1) $\mathcal{C}_i$ chooses 2 random numbers r_i, t_i, computes $w = H(F), c_i = SE(w, r_i), z_i = g^{t_i}$, sends (c_i, z_i) to $\mathcal{S}$.
2) $\mathcal{S}$ forwards $\{c_i, z_i, ID_i\}_n$ to those clients who are in the list $(\{ID_i, C_i\}_n, sh)$.
3) Upon the receipt of $\{c_i, z_i, ID_i\}_n$ from $\mathcal{S}$, each $\mathcal{C}_i$ calculates $X_i = (z_{i+1}/z_{i-1})^{t_i}$ and decrypts $\{c_i\}_n$ to obtain $\{r'_i\}_n$ by using its owned w. Then, X_i is sent to $\mathcal{S}$.
4) $\mathcal{S}$ forwards $\{X_i\}_n$ to $\mathcal{C}_i$ to calculate $sk = Z_{i-1}^{nt_i} \cdot X_i^{n-1} \cdot X_{i+1}^{n-2} \cdots X_{i+n-2} = g^{t_1t_2+t_2t_3+\cdots+t_nt_1}$, $gk = sk \cdot H(w|r'_1|r'_2|\cdots)$, $K_i = SE(gk, \kappa_i)$. Then $(H(gk), K_i)$ is sent to $\mathcal{S}$. $\mathcal{S}$ collects these $\mathcal{C}_i$ who have the same $H(gk)$ to a logic group and maintains another tuple $(\{ID_i, K_i\}, H(gk))$. Note that, the same file F produces the same $H(gk)$ of course. Therefore, $\mathcal{S}$ can determine the logic group by the value of $H(gk)$ which represents a group tag. The duplication check is shown in Fig. 3.

- **File Retrieval**.
The next thing for $\mathcal{S}$ is to preserve one copy of some client, say $\mathcal{C}_j$, and delete the other redundant ciphertexts Cs in this group. For the preserved tuple (ID_j, C, sh), $\mathcal{S}$ sends K_j to another client $\mathcal{C}_i$. Then $\mathcal{C}_i$ uses its gk to decrypt for obtaining κ_j. Once $\mathcal{C}_i$ needs to retrieve the file, it uses κ_j to decrypt C.

4.3 Discussion

4.3.1 Our Scheme v.s. Liu's Scheme

Liu et al. proposed the first *single-server* secure deduplication with encrypted data in CCS 2015. It is resistant to offline brute-force guess attack by a short hash value of the uploading file F as the value of short hash has high collision rate. On the other hand, the 2 clients with the same short hash value perform the same-input-PAKE protocol. Therefore, the 2 clients get the same session key k if they have the same file F. And the interaction of PAKE protocol avoids the offline brute-force guess. However, the same session key is not used as the identical tag of the file directly. Instead, an additively homomorphic encryption scheme is called to transfer $E(H_1(k_F), F)$ to $E(H_1(k_{F_j}), F_j)$ iff $F = F_j$. Hence, the server $\mathcal{S}$ can check the duplication.

However their scheme can not be extended to the batch deduplication. The underlying 2-party PAKE protocol only supports the deduplication one by one. Moreover, the homomorphic encryption increases the interaction steps and computation cost undoubtedly. It does not adapt to large-scale data backup system.

By contrast, our scheme focuses on batch check for the duplicative files in the backup system. Besides, the offline/online guess attacks should be resistant in case of file privacy leakage. Similar to Liu's scheme, we make use of the high collision rate of short hash to avoid using dual servers to prevent offline guess attack. Therefore in the first step, $\mathcal{C}_i$ uploads $(ID_i, SE(\kappa_i, F), sh)$ to $\mathcal{S}$. Semantically secure symmetric encryption scheme SE with unknown encryption key κ_i and short hash sh prevent the adversary's guess on the possible files. Moreover, κ_i may be transit later for others who have the identical files to decrypt. It thus can not be the long-term secret key of the client. Instead, we make use of the onewayness of function H to protect the secret key k_i of the client $\mathcal{C}_i$. Then $\mathcal{S}$ checks the same sh to filter the possible identical files. In the phase of duplication check, we focus on batch check. That is $\mathcal{S}$ determines the identical files from database in constant number of execution. This is the obvious improvement compared to Liu's scheme. Our strategy is to use the key derived from the file to encrypt a random value r_i for each client. Then the encryption is forwarded by $\mathcal{S}$ to other clients. Each one decrypts to obtain these random values by the key derived from its own file. Obviously, the same file results to get the same random values. Therefore, the tag = $H(w|r_1|\cdots)$ can be used to determine the underlying identical files. Hence, $\mathcal{S}$ can check the same encrypted files by collecting these tags and it is not necessary for $\mathcal{S}$ to check the duplication files one by one. In order to prevent $\mathcal{S}$ guessing the files by the given tags offline, we require each client who has the same sh to perform a group key exchange, e.g., BD protocol. The shared group session key blinds the tag, thus $\mathcal{S}$ can not launch such attacks. Moreover, the decryption key κ_i of the encrypted file is encrypted under gk with symmetric encryption, it is easy for the others who have the identical files to obtain.

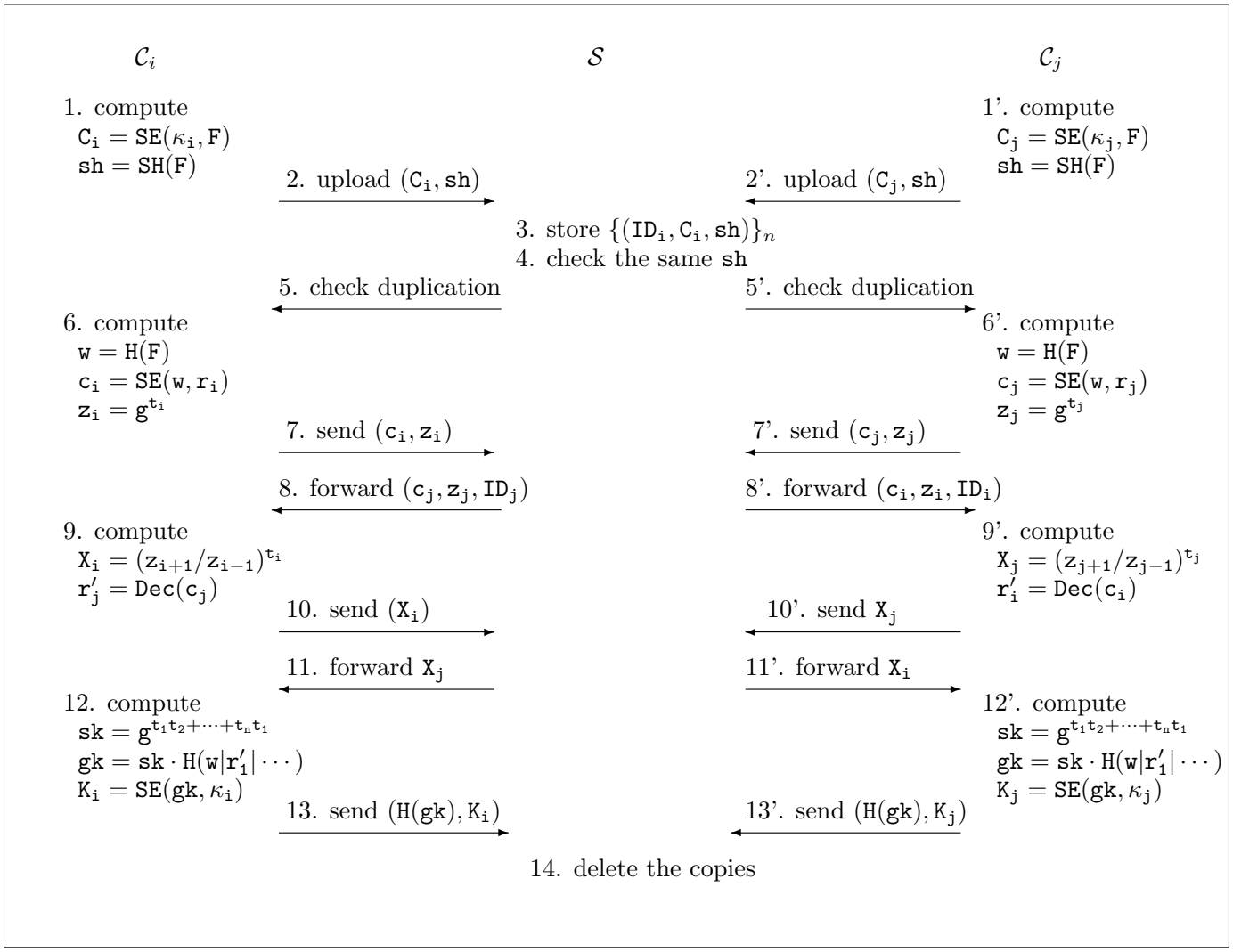

Fig. 3. Duplication Check

4.3.2 *Threshold Selection*

In our scheme, the users who have the same short hash value sh would be invited by S to perform the following algorithm of duplication check. Since the high collision rate of short hash, the same sh does not imply the same file. In other words, several interactions to check duplication are in vain. In order to trade off the communication/computation overhead and storage cost, it is necessary for us to research the threshold value that needs to perform the duplication check. It is obvious to see that if the number of the same sh which is in S's list $(\mathtt{ID_i}, \mathtt{C_i}, \mathtt{sh})$ is too small, say 2, it will cause lots of unnecessary communication overhead. If the selected threshold value is too big, it will cause unchecked duplication. Therefore, we implement our scheme in different thresholds to show our result. It will be elaborated in Section 6.

4.4 Comparative Analysis

We analyze the efficiency of the proposed scheme in this section. TABLE 2 shows the comparison of our scheme and some typical schemes [10], [17], [19], [20], [39], [38].

Among deduplication schemes, CE or MLE has the lowest computational cost because they do not have complex mathematical operations like group multiplication operation and exponentiation to perform file encryption. However, due to the key space is so small, it can not prevent brute-force attack. Therefore, the outsourced data confidentiality can not be guaranteed. There are also some other encrypted deduplication without the aid of additional servers like Shin [38] and Abadi [37]. Abidi proposed a full randomized all components of the ciphertexts which is based on pairing operation. Shin's solution is also based on a bilinear pairing operation to test whether the files are identical. Obviously, it can only check the duplication one by one and bilinear pairing operation is very time-consuming.

Under the aid of additional servers, some schemes enable resistance to brute-force attacks by using OPRF and other variants. The encryption key is derived from the file F and an additional global secret which is from servers. Therefore, the key is independent of the file content. But these schemes suffered from server-compromise attacks. Miao [18] and Duan [19] try to mitigate this problem by employing multiple key servers. If more than k servers (i.e.,

TABLE 2
Comparison of deduplication schemes

Test Groups	MLE(CE) [10]	DupLESS [17]	Liu et al. [20]	SED [39]	Shin et al. [38]	Our Scheme
Tag Computation	*H*	*H*	*H + SE*	*3E*	*E*	*SH*
Key Generation	*H*	*H + 2M + 2E*	*H*	*H*	*H + E*	*H*
File Encryption	*SE*	*SE*	*SE*	*SE + M + P + 2Xor*	*SE + 3E + P*	*H + SE*
Server-aided Requirement	✖	✔	✖	✔	✖	✖
Brute-force Attack Resistance	✖	✔	✔	✔	✔	✔
Server-compromise Attack Resistance	✖	✖	✔	✖	✔	✔
Batch Checking	✖	✖	✖	✖	✖	✔

H: hash operation, SE: symmetric encryption, M: group multiplication, E: group exponentiation, P: pairing operation, k: number of key servers.

threshold) servers are compromised, then these approaches will still fail. In addition, the security of these schemes wiil reduce to that of the original MLE if the global secret is leaked from servers. Moreover, the cost of employing multiple key servers is very expensive. The interaction between the client and the key servers introduces significant latency and communication overhead before the data is encrypted and uploaded to the server.

5 Security Analysis

According to the attack model presented in Sec. 3.3, our system mainly focuses on *Online Brute-force Attack* and *Offline Brute-force Attack* by the malicious server $\mathcal{S}$.

Intuitively, our scheme is against online brute-force attacks by limiting the number of **Deduplication Check** runs. Indeed, the algorithm of deduplication check is *interactive*. Although it is communication consuming and online for both sides required, the restricted runs can improve the security to some extent and reduce the overhead both, i.e., the malicious server $\mathcal{S}$ pretends to one of the uploader to run with other uploaders who have the same short hash value $\mathtt{sh}$ for predictable files.

Generally, our scheme is against offline-force attacks without dual servers aid. Firstly, the file uploaded to the server is encrypted by a secure encryption with a random key. It is impossible for $\mathcal{S}$ to guess the underlying file $\mathtt{F}$ as κ_i is random. Then, the short hash $\mathtt{sh}$ stored in $\mathcal{S}$ avoids guessing the content of $\mathtt{F}$ offline by $\mathcal{S}$. Due to the high collision rate of $\mathtt{SH}()$, a same short hash $\mathtt{sh}$ may be associated with different encrypted files whose plaintext maps to $\mathtt{sh}$. BD protocol (group key agreement protocol) is performed to generate a shared secret $\mathtt{sk}$ to blind the value of $\mathtt{H}(\mathtt{w}|\mathtt{r}'_1|\cdots)$. It prevents a curious server $\mathcal{S}$ to predict a file by computing the right $\mathtt{r_i}$ offline.

In order to prove our scheme presented in Sec. 4.2 is secure against malicious central server $\mathcal{S}$, we follow the work [20] to show that the execution of the deduplication in the real model is computationally indistinguishable from the execution in the ideal model.

Ideal Model. The deduplication protocol can be modeled as two roles: the central backup server $\mathcal{S}$ and the clients (local servers $\mathcal{C}_i$). $\mathcal{C}_i$ upload their encrypted files to $\mathcal{S}$ and $\mathcal{S}$ interacts with $\mathcal{C}_i$ to check the duplication. Specifically, $\mathcal{S}$ has no information about $\mathcal{C}_i$'s files although these files may be predictable. The ideal functionality of deduplicating encrypted files is defined in Fig. 4.

Real Model. However, an adversary $\mathcal{A}$ may compromise $\mathcal{S}$ in the real model, say the malicious server $\mathcal{S}'$. The compromised $\mathcal{S}'$ may masquerade as one client $\mathcal{C}_i$ to check the files stored in it by observing which one causes deduplication.

Input:

- Uploading: Each $\mathcal{C}_i$ input $\mathtt{F_i}$ and $\mathtt{k_i}$.
- Duplication Check: Each $\mathcal{C}_i$ with the same short hash $\mathtt{sh}$ input $\mathtt{F_i}$, random numbers $(\mathtt{r_i}, \mathtt{t_i})$.
- $\mathcal{S}$'s input in the two phases are empty.

Output:

- Each $\mathcal{C}_i$ with the same short hash $\mathtt{sh}$ gets its own $\mathtt{gk}$. If the underlying file $\mathtt{F_i}$ are identical, these $\mathtt{gk}$ are identical.
- Uploading: $\mathcal{S}$ gets the cipertext $\mathtt{SE}(\mathtt{k_i}, \mathtt{F_i})$ for $\mathcal{C}_i$.
- Duplication Check: $\mathcal{S}$ gets $(\{\mathtt{ID_i}, \mathtt{k_i}\}, \mathtt{H}(\mathtt{gk}))$ for a logic group which has the same file.

Fig. 4. The Ideal Functionally of Deduplicating Encrypted File

Theorem 1. *The deduplication protocol presented in Sec. 4.2 is secure against malicious server $\mathcal{S}'$ if the BD protocol is a secure group key agreement and* $\mathtt{SE}$ *is a semantically secure symmetric encryption scheme.*

Proof. The proof idea is to show that the execution of the deduplication protocol in the real model is computationally indistinguishable from the ideal model. Therefore, we construct a simulator $\mathcal{SIM}$ that can both access our deduplication oracle in ideal model and obtain messages the compromised $\mathcal{S}'$ would send in real model. $\mathcal{SIM}$ generates the communication transcript of the real model execution that is computationally indistinguishable from that of the ideal model execution. Similar to [20], we assume that the underlying BD group key agreement protocol is implemented as an oracle the parties can quire.

We construct the simulator $\mathcal{SIM}$ for the compromised server $\mathcal{S}'$. $\mathcal{SIM}$ now plays the role of the clients $\mathcal{C}_i$ to send short hash values $\mathtt{sh_i}$ to $\mathcal{S}'$. $\mathcal{S}'$ collects the clients $\mathcal{C}_i$ with the same $\mathtt{sh_i}$. Then these $\{\mathcal{C}_i\}$ perform the duplication check protocol and $\mathcal{S}'$ forwards the messages produced by $\mathcal{C}_i$ in this protocol. $\mathcal{SIM}$ continues to pretend to be these $\{\mathcal{C}_i\}$ to generate $(\mathtt{c_i}, \mathtt{z_i}, \mathtt{X_i})$ by randomly chosen $\mathtt{w}, \mathtt{r_i}, \mathtt{t_i}$. Once upon the receipt of the encryption $\mathtt{c_i}$ and $\mathtt{X_i}$ from $\mathcal{S}'$, $\mathcal{SIM}$ picks $\mathtt{r_i}$ chosen before and sets $\mathtt{gk}$ as a random value $\mathtt{x}$ in the ideal model. Then $\mathcal{SIM}$ sends $(\mathtt{H}(\mathtt{x}), \mathtt{K_i})$ to $\mathcal{S}'$.

The view of $\mathcal{S}'$ in the real model is $view^{\mathcal{S}'}_{REAL} = < \mathtt{SE}(\kappa_i, \mathtt{F}), \mathtt{sh}, \mathtt{H}(\mathtt{sk} \cdot \mathtt{H}(\mathtt{w}|\mathtt{r_1}|\cdots)), \mathtt{K_i} >$ and the view of

$\mathcal{S}'$ in the ideal model is $view^{\mathcal{S}'}_{IDEAL} = < \widetilde{C}, \mathtt{sh}, \mathtt{H(x)}, \mathtt{K_i} >$ where $\widetilde{C}$ and $\mathtt{x}$ are random. We show that the distributions of $view^{\mathcal{S}'}_{REAL}$ and $view^{\mathcal{S}'}_{IDEAL}$ are identical. Since SE is semantically secure encryption algorithm and F is encrypted by a randomly generated key κ_i, the ciphertext of F is indistinguishable from a random value $\widetilde{C}$. Meanwhile, sk is the output of a secure group key agreement, it is a random group key output by the BD protocol. Therefore, $\mathcal{S}'$ cannot distinguish $\mathtt{sk} \cdot \mathtt{H(w|r_1|} \cdots \mathtt{)}$ in the real model from a random x in the ideal model. Thus we can see that $view^{\mathcal{S}'}_{REAL}$ and $view^{\mathcal{S}'}_{IDEAL}$ are identically distributed, which means that $\mathcal{S}'$ cannot violate the security of deduplication protocol. □

6 PERFORMANCE EVALUATION

We implement our scheme by C++ language and test its performance on a virtual machine using Ubuntu version 20.04 with an Intel Core i5-9300H, CPU @ 2.40GHz, and 4G DRAM. We use OpenSSL (version 1.1.1f) to implement MD5 as the cryptographic hash function and set the length of short hash n = 16 bits. We used AES as the symmetric encryption scheme. We did not take the time of file uploading and downloading into consideration. For performance tests, we present the average results over 20 runs.

6.1 File Encryption and Decryption

In the experiment, we first tested the time of file encryption and decryption if the SE algorithm is an AES initialization. We used key with different size (128 bits, 196 bits and 256 bits) to operate different file size. As shown in Fig. 5 and Fig. 6, we noted that even when using the 256-bit key to operate 600Mb file, the decryption/encryption time is still less than 9 seconds. We acknowledge that the decryption/encryption time almost increases linearly with the increase of the file size. This is unavoidable in all symmetric encryption schemes. AES is very efficient and practical to be applied for encrypted data deduplication. Computational overhead includes one AES encryption and one hash operation. We also tested the performance of MD5 for tag generation as shown in Fig. 7. We note that the delay caused by the hash operation is a small fraction of the overall file operation process. File encryption accounts for a large portion of the computational overhead on the user side. Therefore, we also tested file encryption time for several schemes. The results are shown in Fig. 8. The y-axias refers to the time taken by the client to calculate the encrypted file C from the corresponding file F. The comparisons are as shown in Fig. 8. We find that the time spent implementing file encryption of all schemes increases with the size of files increasing. However, our scheme is more efficient than the previous schemes. Bellare et al.'s scheme [17] require large computations with remote entity during file encryption.

6.2 File Deduplication

6.2.1 Dataset

In backup storage systems, workloads usually are a series of backups versions. Thus, backup systems have high duplicate ratio. For our experiment, it is not easy to obtain a large volume of real-world backup files. To investigate the

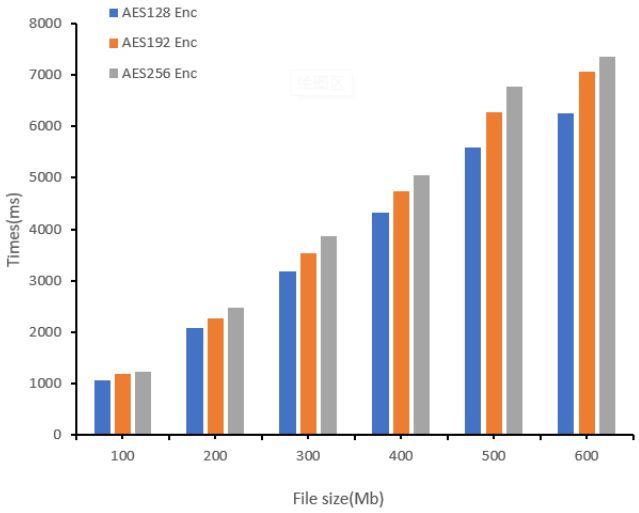

Fig. 5. File encryption time with AES

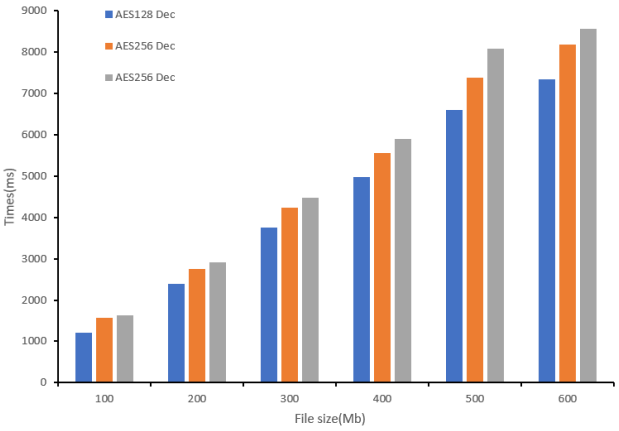

Fig. 6. File decryption time with AES

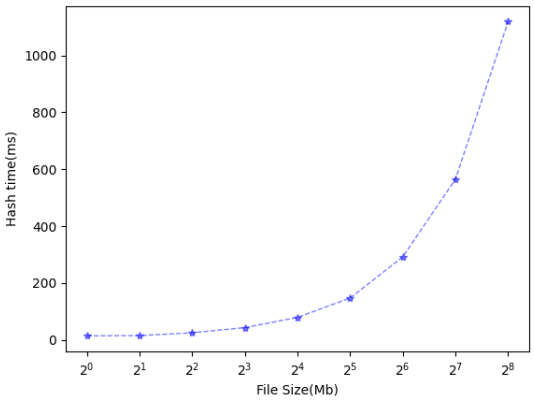

Fig. 7. Hash operation time with MD5

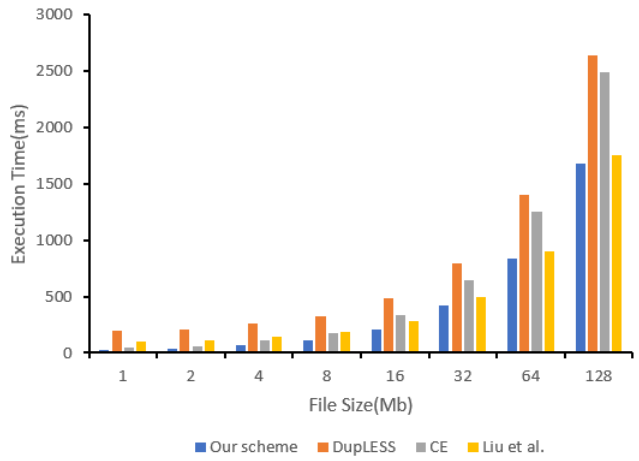

Fig. 8. File encryption for different schemes

performance of our proposed batch deduplication scheme, we synthesized three 600Mb datasets with different duplicate ratios (70%, 80%, 90%) before deduplication.

6.2.2 Parameters

The short hash used in our experiment were implemented by truncating the hash values of the encrypted data and we set the length = 16 bits. Morever, to save bandwidth, we set up different thresholds. If the number of logic group members is less than the threshold, no negotiation is performed. We use these parameters in our simulations.

Our experiment was inspired by the *map* and *set* primitives present in many functional languages. We realized that most of our computations involved applying a *unordered map* operation to each logical record in server's input in order to compute a lot of intermediate key/value pairs. Then we applied a *set* operation to all the values that share the same key, in order to store different encrypted values. The use of a functional model allow us to parallelize large computation easily. In our scheme, grouping and key negotiation are the most time-consuming operations in deduplication phase. Therefore, our work mainly makes it more applicable in cloud storage. Fortunately, our scheme supports parallel execution of the grouping and key negotiation which makes our scheme outperform other deduplication schemes [17] [18] [19]. Our scheme requires only a simple upload of encrypted files. Liu's scheme, on the other hand, requires a series of interactive negotiations before uploading the file. This scheme is hard to achieve in practical applications

The experiment was conducted as follows. We simulated users encrypted the files in the dataset and calculated the short hash of all the files. Then they sent them to the server. For a file, we created a mapping between its short hash value and the encrypted value in the server. We checked the map and considered files with the same short hash value as a group. In each group, we executed batch duplication check with different thresholds and group key agreement protocol. After that, we use *set* to store the deduplicated files values. By using *map* and *set* primitives, we can and rapidly executed the batch checking and accurately computed the duplicate ratio.

6.2.3 Threshold Selection

There is a dilemma between deduplication efficiency and communication cost. Obviously, if the selected threshold value is too small, it will cause a lot of unnecessary communication overhead. That is because the identical short hashes do not mean the same files. If the selected threshold value is too big, there will many identical files that are not checked. Therefore, in our simulation, we set up different thresholds and test the performance. The efficiency of deduplication has different results for different threshold values. The batch deduplication effect is shown in Fig.9. Our proposed scheme is implemented with a 90% repetition ratio dataset. When the threshold value is 2, the duplicate ratio of the dataset after deduplication is only 0.16%, and the duplicate ratio after de-duplication are 1.49%, 5.60%, and 17.30% when the threshold values are 3, 4 and 5, respectively. When the threshold is set to 2, the communication overhead reaches its maximum. With a 90% duplicate ratio dataset, when the threshold is 5, communication overhead decreases by 2.74% compared to the threshold is 2. However, with an 80% duplicate ratio dataset, communication overhead decreases by 32.83% compared to the threshold is 2.

Therefore, we suggest that when the backup system has a high duplicate ratio, users can lower the threshold value to save more storage space, and when the repetition rate is low, the system can make a trade-off based on the actual situation. In addition, in our scheme, communication overhead is independent of file size. Therefore, when the file is large enough, the overhead is negligible.

The results demonstrate that our scheme can save a mass of storage space in small threshold and very efficient to be applied in real-world backup systems.

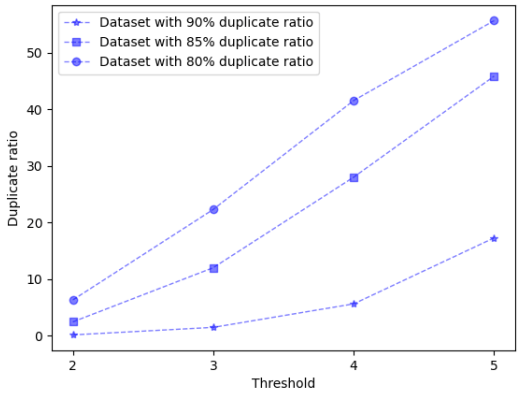

Fig. 9. Deduplication percentage with different thresholds

6.3 Efficiency Of Duplicate File Checking

In this section, we implemented some other deduplication schemes [17] [20] and compare the efficiency between them. We choose [17] because it has the lowest computational overhead among the server-aided schemes. Our scheme focuses on the security and efficiency of deduplication, thus

we test whether our scheme is effective in detecting duplicates, rather than the speed of eliminating them. Therefore, we select a different number of files from the dataset to perform file-level deduplication. Because our scheme supports batch checking, as shown in Fig.10, we find that as the number of files increases, the time taken to implement deduplication increases for all schemes. However, the growth rate of our scheme is the slowest. Because the probability of short hash collisions rises as the number of files increases, batch checking is faster than other schemes. In summary, our scheme outperforms other previous schemes in terms of the efficiency of duplicate data checking.

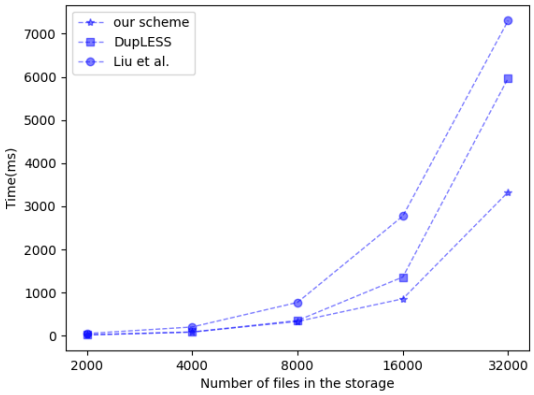

Fig. 10. The running time for different amounts of files.

7 Conclusion

In this paper, we proposed a novel semantic secure encrypted deduplication scheme without dual servers that supports batch checking for underlying identical files. By using the high collision rate of the short hash, it not only singles out the underlying identical files, but also prevents offline guesses by the central server. Each local server, i.e., the client, needs to interact with other clients to check whether their files are the same. By limiting the number of protocol instances, it prevents online guesses. The parallel calculations make the central server to check whether the massive encrypted files are the same or not in constant executions.

We extensively evaluate batch deduplication from file encryption, hash operation, and storage efficiency aspects. We show that our scheme significantly mitigates the encrypted data storage overhead in cloud storage especially in backup systems.

Acknowledgment

This work is supported by Chengdu Science and Technology Program (2021-YF08-00151-GX) and Sichuan Science and Technology Program (2019YFG0508, 2020JDTD0007).

References

[1] IDC. Data age 2025. https://www.seagate.com/our-story/data-age-2025/.
[2] D. T. Meyer and W. J. Bolosky, A study of practical deduplication, ACM Trans. Storage, vol. 7, no. 4, pp. 1-20, 2012.
[3] G. Wallace, F. Douglis, H. Qian, P. Shilane, S. Smaldone, M. Chamness, et al., Characteristics of Backup Workloads in Production Systems,. In Proc. 10th USENIX Conf. FAST, pp. 1-16, 2012.
[4] GoogleDrive. [Online]. Available: http://drive.google.com (2012).
[5] Dropbox. [Online]. Available: http://www.dropbox.com (2007).
[6] Memopal. [Online]. Available: http://www.memopal.com (2018).
[7] D. Harnik, B. Pinkas and A. Shulman-Peleg, Side Channels in Cloud Services: Deduplication in Cloud Storage. In IEEE Security & Privacy, vol. 8, no. 6, pp. 40-47, 2010.
[8] P. Puzio, R. Molva, M. Onen and S. Loureiro, ClouDedup: Secure deduplication with encrypted data for cloud storage. In Proc. IEEE Int. Cof. Cloud Comput. Technol. Sci., pp. 363-370, 2013.
[9] J. Li, C. Qin, P. P. Lee and X. Zhang, Information leakage in encrypted deduplication via frequency analysis. In Proc. 47th Annu. IEEE/IFIP Int. Conf. Depend. Syst. Netw., pp. 1-12, 2017.
[10] J. R. Douceur, A. Adya, W. J. Bolosky, D. Simon and M. Theimer, Reclaiming space from deplicate files in a serverless distributed file ststem. In Proc. IEEE Int. Conf. Distrib. Comput. Syst., pp. 617-624, 2002.
[11] M. Bellare, S. Keelveedhi, and T. Ristenpart, Message-locked encryption and secure deduplication. In Proc. 32nd Annu. Int. Conf. Theory Appl. Cryptographic Techn., pp. 296-312, 2013.
[12] Y. Zhang, C. Xu, H. Li, K. Yang, J. Zhou and X. Lin, Healthdep: An efficient and secure deduplication scheme for cloud-assisted ehealth systems. In IEEE Transactions on Industrial Informatics, vol. 14, no. 9, pp. 4101-4112, 2018.
[13] F. Armknecht, C. Boyd, G. T. Davies and Gjsteen, Side channels in deduplication: Trade-offs between leakage and efficiency. In Proc. ACM Conf. Comput. Commun. Security, pp. 266-274, 2017.
[14] H. Cui, C. Wang, Y. Hua, Y. Du and X. Yuan, A Bandwidth-Efficient Middleware for Encrypted Deduplication. In Proc. of IEEE DSC, 2018.
[15] S. Halevi, D. Harnik, B. Pinkas and A. Shulman-Peleg, Proofs of ownership in remote storage systems. In Proc. 18th ACM Conf. Comput. Commun. Secur., pp. 491-500, 2011.
[16] D. Harnik, B. Pinkas, and A. Shulman-Peleg. Side channels in cloud services: Deduplication in cloud storage. In IEEE Security & Privacy, 8(6):40-47, 2010.
[17] M. Bellare, S. Keelveedhi, and T. Ristenpart. DupLESS: Server-aided encryption for deduplicated storage. In Proc. of USENIX Security, 2013.
[18] M. Miao, J. Wang, H. Li, and X. Chen, Secure multi-server-aided data deduplication, cloud computing, Pervasive Mobile Comput., vol. 24, pp. 129-137, 2015.
[19] Y. Duan, Distributed key generation for encrypted deduplication achieving the strongest privacy. In Proc. 6th Ed. ACM Workshop Cloud Comput. Security, pp. 57-68, 2014.
[20] J. Liu, N. Asokan, and B. Pinkas, Secure deduplication of encrypted data without additional independent servers. In Proc. 22th ACM SIGSAC Conf. Comput. Commun. Secur., pp. 874-885, 2015.
[21] S. M. Bellovin and M. Merritt, Encrypted key exchange: Password-based protocols secure against dictionary attacks. In Proc. IEEE Security & Privacy, pp. 72-84, 1992.
[22] M. Naor and O. Reingold. Number-theoretic constructions of efficient pseudo-random functions. In Journal of the ACM, 51(2):231-262, 2004.
[23] J. Li, Z. Yang, Y. Ren, P. Lee, and X. Zhang. Balancing storage efficiency and data confidentiality with tunable encrypted deduplication. In Proc. of ACM Eurosys, 2020.
[24] Y. Zhang, C. Xu, N. Cheng and X. Shen, Secure password-protected encryption key for deduplicated cloud storage systems. In IEEE Transactions Dependable and Secure Computing, pp. 1-18, 2021.
[25] H. Yuan, X. Chen, J. Li, T. Jiang, J. Wang, and R. Deng, Secure Cloud Data Deduplication with Efficient Re-encryption. In IEEE Transactions on Services Computing, 2019.
[26] J. Li, C. Qin, P. P. C. Lee and J. Li, Rekeying for encrypted deduplication storage. In Proc. 46th Annu. IEEE/IFIP Int. Conf. Depend. Syst. Netw., pp. 618-629, 2016.
[27] H. Cui, R. H. Deng, Y. Li, and G. Wu, Attribute-based storage supporting secure deduplication of encrypted data in cloud. In IEEE Transactions on Big Data, vol. 5, no. 3, pp. 330-342, Sep. 2019.
[28] M. Li, C. Qin and P. P. Lee, CDStore: Toward reliable secure and cost-efficient cloud storage via convergent dispersal. In Proc. USENIX Conf. Usenix Annu. Tech. Conf., pp. 111-124, 2015.

[29] P. Anderson and L. Zhang, Fast and Secure Laptop Backups with Encrypted De-Duplication. In Proc. USENIX LISA, pp. 1-8, 2010.
[30] M. Bellare and S. Keelveedhi, Interactive message-locked encryption and secure deduplication. In Public-Key Cryptography, pp. 516-538, 2015.
[31] M. W. Storer, K. Greenan, D. D. E. Long and E. L. Miller, Secure data deduplication. In Proceedings of 4th ACM International Workshop Storage Security Survivability, pp. 1-10, 2008.
[32] Y. Zhang, C. Xu, N. Cheng, and X. Shen, Secure Encrypted Data Deduplication for Cloud Storage against Compromised Key Servers. In IEEE Global Communications Conference, pp. 1-6, 2019.
[33] Y. Ren, J. Li, Z. Yang, P. P. C. Lee and X. Zhang, Accelerating encrypted deduplication via SGX. In Proc. of USENIX ATC, 2021.
[34] Li, C. Qin, P. P. C. Lee, and X. Zhang. Information leakage in encrypted deduplication via frequency analysis. In Proc. of IEEE/IFIP DSN, 2017.
[35] M. Burmester and Y. Desmedt, A secure and efficient conference key distribution system.In Proc. Workshop Theory Appl. Cryptograph. Techn., pp. 275-286, May 1994.
[36] J. Ma, R. J. Stones, Y. Ma, J. Wang, J. Ren, G. Wang, et al., Lazy exact deduplication. In ACM Transactions on Storage, vol. 13, no. 2, pp. 11, 2017.
[37] M. Abadi, D. Boneh, I. Mironov, A. Raghunathan, and G. Segev, Message-locked encryption for lock-dependent messages, Advances in Cryptology-CRYPTO 2013, vol. 8042, pp. 374-391, 2013.
[38] Y. Shin, D. Koo, J. Yun, and J. Hur, Decentralized server-aided encryption for secure deduplication in cloud storage. In IEEE Transactions on Services Computing, vol. 13, no. 6, pp. 1021-1033, 2020.
[39] D. Zhang, J. Le. Secure and Efficient Data Deduplication in Joint-Cloud Storage. In IEEE Transactions on Cloud Computing. 2021.