Data Splitting based Double Layer Encryption for Secure Ciphertext Deduplication in Cloud Storage

Xin Tang, Luchao Jin
School of Cyber Science and Engineering, University of International Relations
Beijing, China
xtang@uir.edu.cn, jin_785515@uir.edu.cn

2024 IEEE 17th International Conference on Cloud Computing (CLOUD) | 979-8-3503-6853-6/24/$31.00 ©2024 IEEE | DOI: 10.1109/CLOUD62652.2024.00027

***Abstract*—Ciphertext deduplication is an emerging technique to eliminate redundancy in cloud storage with data confidentiality guaranteed. However, the encryption key has to be synchronized among different owners of the same data, which in turn makes the existing deduplication schemes suffer from security problems such as brute-force attacks and the collusion attack. In order to deal with them in a lightweight way, we propose a data splitting based double layer encryption for secure ciphertext deduplication in cloud storage, which is the first work to solve the problem of privacy exposure at a fundamental level. Specifically, we propose a bloom filter based data splitting method and a double layer encryption based deduplication framework. Because of the high collisional property of the feature extraction method, the same content could be extracted from similar data with a great probability. By encrypting it with a shared key, and the remaining part with the secret key of the uploader, the existence privacy is well protected during the process of ciphertext deduplication as a result. According to the experimental results compared with the state-of-the art, the communication and storage overhead can be reduced over 82.28% and 5.55% under the proposed scheme, which shows that secure ciphertext deduplication can be achieved in a lightweight way.**

***Index Terms*—Ciphertext deduplication, cloud storage, brute-force attack, collusion attack, existence privacy**

I. Introduction

With the arrival of the big data era, the volume of data increases explosively on the Internet. According to a survey done by IDC (International Data Corporation) [1], the total amount of data almost doubles every two years, which would reach 175 zettabytes by 2025. Under such a considerable growth rate, how to achieve efficient storage and management of data is becoming a big challenge [2]–[4]. Cloud storage has been widely employed as an effective means to deal with the challenge, by enabling users to outsource their data before deleting from local storage. In order to further eliminate storage and communication overhead, cross-user data deduplication [5]–[11] is usually supported by commercial cloud service providers (CSP). To protect the confidentiality from being leaked, the data is usually encrypted before uploading. However, such a manner actually brings about a challenge to deduplication since even though the plaintext data is identical, the ciphertext cannot be simply deduplicated because they are actually encrypted by different keys. Therefore, how to share the encryption key among legitimate data owners is an urgent problem that needs to be solved.

Deterministic encryption [12]–[22] is an effective technique in dealing with this problem, in which an identical encryption key can be derived from a certain plaintext to achieve deduplication. However, because of the deterministic nature of such a manner, the key could also be inferred by an adversary since the content of data with low min-entropy is easy to be disclosed through brute-force attacks [23]. Even though this problem can be solved by introducing randomness through an independent key server [14], such a solution suffers from single point of failure problem [21] and the sybil attack [24]. Once the server is invalidated or compromised, or the user's identity is illegally forged, the randomness introduced can be easily obtained. As a result, the deterministic nature is actually inherited. On the other hand, non-deterministic encryption [23], [25]–[28] is proposed by employing a randomly generated key instead. To achieve cross-user deduplication, the key has to be transferred among legitimate owners of the same data. Building secure channels [28] and adopting homomorphic encryption [23] are mainstream methods to achieve this goal, even though effective to some extent, they either rely on inefficient multi-party interactions which occupy a large amount of bandwidth or still suffer from brute-force attacks. Moreover, consider a more complicated scenario that the malicious CSP collaborates with some users to launch collusion attack, none of the solutions mentioned above is effective any more. The fundamental reason can be attributed to the one-to-one correspondence between the encryption key and the plaintext to be uploaded. In other words, a randomly generated key is uniquely utilized to encrypt the given data. Once it is compromised by adversaries, the confidentiality is broken instantly in fact.

In this context, attempt to solve the aforementioned security problems in ciphertext deduplication at a fundamental level with efficiency guaranteed, we propose a data splitting based double layer encryption (DSDE) for secure ciphertext deduplication in cloud storage without third-party involvement. To the best of our knowledge, DSDE is the first work to well resist brute-force attacks and the collusion attack in ciphertext deduplication in a lightweight way. Specifically, we propose a bloom filter based data splitting strategy, ensuring the same content could be extracted from similar data with a great probability. By encrypting it with a shared key, and the remaining part with the secret key of the uploader, the existence privacy is well protected during the process of

2159-6190/24/$31.00 ©2024 IEEE
DOI 10.1109/CLOUD62652.2024.00027

ciphertext deduplication as a result. Our main contributions are summarized as follows:

i. We put forward a novel ciphertext deduplication framework, which supports data splitting and double layer encryption. With the help of this framework, the one-to-one correspondence between the encryption key and the plaintext to be uploaded is broken, thus even if the key is exposed through brute-force attacks and the more sophisticated collusion attack, the existence privacy of the target data is still well protected during the process of ciphertext deduplication.
ii. Next, we focus on designing a bloom filter based method to achieve data splitting, ensuring the same content could be extracted from similar data with a great probability. By performing the proposed homomorphic encryption based lightweight key transferring, privacy-preserving secure ciphertext deduplication could be achieved in an efficient way.
iii. We perform security analysis for DSDE and take experiments on real-word datasets to evaluate the performance. Both theoretical and experimental results show that, DSDE is able to achieve efficient ciphertext deduplication with resistance against brute-force attacks and the collusion attack simultaneously.

II. Related Works

In this section, we present an overview of the mainstreamed solutions to achieve cross-user deduplication for ciphertext and point out their limitations. In particular, according to the method of key generation, the techniques discussed are classified into deterministic and non-deterministic encryption oriented ones respectively.

Deterministic encryption works by producing a fixed encryption key for a given plaintext. By executing encryption with the help of the key, a certain plaintext will result in a consistent ciphertext, thus enabling cross-user ciphertext deduplication in a simple way. The first solution to achieve deterministic encryption is convergent encryption (CE) proposed by Douceur *et al.* [12], in which hash value of the file to be deduplicated is used as the encryption key. Following their work, message-locked encryption (MLE) [13] formalizes CE with formal security definitions. However, the key involved in [12] and [13] are essentially derived from the plaintext itself, which is inevitably vulnerable to offline brute-force attack. Under such an attack, the malicious CSP is completely able to generate multiple versions of the target file by traversing a pre-determined dictionary, before deriving their corresponding keys. Once the obtained ciphertext is existent in cloud storage, the existence privacy of the target file is exposed instantly. In order to address this attack, server-aided MLE [14] was proposed, where a global randomness is introduced into the process of key generation by utilizing an independent key server. Nevertheless, such a solution is fragile against single point of failure. If the key server is compromised, the introduced randomness is able to be obtained by an adversary, thus breaking the security. Although multi-server-aided MLE [19]–[21] was presented to disperse the risk, independent key servers used by [14], [19]–[21] are inherently unable to circumvent the problem of online brute-force attack, since the randomness can easily be obtained by a malicious CSP through identity forgery attack [24].

Considering the problems mentioned above, non-deterministic encryption is proposed by utilizing a randomly generated key instead. In this solution, how to transfer the encryption key becomes a big challenge. Identity-based broadcast encryption (IBBE) was employed by Bai *et al.* [25] and Liu *et al.* [26] to achieve key transferring. According to their design, the key randomly generated by the first uploader is encrypted by another key associated with a pre-determined group. Users in the group are able to derive the key with their respective identity IDs. Based on these works, Yuan *et al.* [27] proposed a ciphertext-policy attribute-based encryption (CP-ABE) where the attributes of the user are used as keys to encrypt the selected random key. As a result, users with the specific attributes are allowed to access the key. However, in these schemes, key transferring is only allowed within the pre-determined group, which largely restrains the efficiency of deduplication.

To improve the efficiency, Liu *et al.* [28] proposed a scheme based on the same-input-PAKE protocol to break the limitation. According to their design, the randomly generated encryption keys are kept by data owners in local storage. The user who requested to upload is required to communicate with a set of potential online owners specified by the cloud. Only when both parties have interacted with the same hash value, a consistent session key can be derived and the process of key transferring is terminated. In order to lower the risk of key exposure under online brute-force attack, rate limiting is introduced to limit the number of interactions. However, during the process of key transferring, heavyweight communication and storage overhead is inevitable, which in turn brings about a heavy burden to users. Focus on this problem, Pooranian *et al.* [23] resorted to a homomorphic encryption based manner to achieve key transferring. In particular, the keys are stored in cloud storage after been encrypted by a specific homomorphic encryption algorithm. The design of their solution enables the user with consistent hash value of a file to obtain the homomorphic encrypted key. By performing decryption to the key, the corresponding file is able to be encrypted into the same ciphertext so that deduplication can be triggered. However, the scheme is fragile against offline brute-force attack since once hash value of the target file is correctly inferred, the key is exposed instantly. Meanwhile, it suffers from side channel attack because once key generation is not required according to deduplication response, the existence privacy is also exposed.

During the process of deduplication, the index which is usually defined as the hash value of the file [12]–[22], [25]–[27], is also vulnerable to offline brute-force attack especially for predictable data. Although [23], [28] try to utilize short hash values instead to prevent such an attack, due to the avalanche effect of the hash function [29], the effect of resistance is still limited. Furthermore, the schemes introduced

above suffer from the collusion attack [30], where a malicious CSP may collaborate with one or more user to violate the privacy of other ones. Under such an attack, the confidentiality in the cloud can be breached by obtaining the shared key from the conspired users. On the other hand, the CSP is also able to send a certain ciphertext to the conspired one who is in possession of the shared key but is not the legitimate owner.

III. Problem Statement

A. System Model

As shown in Fig. 1, we consider a general cross-user ciphertext deduplication model that contains two kinds of entities: the CSP and the cloud user. We assume that the message/data between different entities are transmitted through secure channels (e.g., using secure socket layer (SSL)/ transport layer security (TLS) protocol). Hence, the attacker cannot monitor communication by employing network package capturing tools any more.

- CSP: The CSP is an entity who provides data storage and computing service for multiple users. In order to minimize both the communication and storage overhead, the CSP performs cross-user ciphertext deduplication to prevent redundant uploading of the identical data even though it is encrypted by different users. According to our design, the CSP is also responsible for transferring encryption keys among different users to support further ciphertext deduplication.
- Cloud user: A cloud user is an entity who outsources data to the cloud to alleviate the burden of its local storage. Taking the confidentiality into account, the cloud user usually encrypts its data before outsourcing. In particular, if the user is the first uploader of a certain file, it has to randomly generate a key before performing encryption. Otherwise, for subsequent uploaders, they have to interact with the CSP to achieve key acquisition before outsourcing.

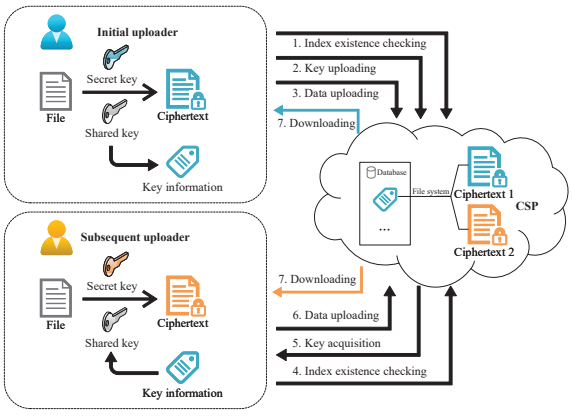


Fig. 1. The system model.

B. Threat Model

Internal Adversary. We consider the threat first comes from the internal adversary. A malicious CSP wants to breach the confidentiality of data in its local storage. For predictable data with low min-entropy, the adversary launches an offline brute-force attack with the help of a pre-determined dictionary. In particular, such an attack is possible in the following two scenarios.

i. Index collision. Once the content of data is inferred according to the aforementioned offline brute-force attack, the corresponding index could be obtained as well since it is usually defined to be the hash value. As a result, if the index of a certain version traversed is consistent with that of the target data in cloud storage, the existence privacy is exposed instantly. Even though the security could be ensured by protecting the index with a randomly generated key. The key is also at risk in the following scenario.
ii. Key acquisition. In order to transfer keys between different data owners, the key related auxiliary information should be stored in cloud storage as well. With the help of this information, the key could be derived by the subsequent owners in the process of key acquisition. However, once the content of the target data is correctly generated by the malicious CSP, the key related information could be derived as well. Thus the key can be obtained. By comparing the generated ciphertext with those in cloud storage, the correctness of the key obtained can be verified at last.

External Adversary. The external adversary could be a malicious cloud user who wants to infer the content of the predictable target data by interacting with the CSP. A side channel is established during the process of deduplication, which could be utilized to steal existence privacy of the target data. In particular, a malicious cloud user is able to construct deduplication requests for each possible version of the target data by launching brute-force dictionary attack. By outsourcing them to the CSP in sequence, the adversary is able to infer which one of the versions is consistent with the target data by observing the returned response. As a result, the existence privacy of the target data is exposed instantly.

Even though the collisional property is introduced in the generation of deduplication index to solve the side channel attack mentioned above, the system still suffers from a more sophisticated attack launched in a collusive way. In particular, such an attack is possible in the following two scenarios.

i. The CSP reveals a certain ciphertext together with its key related auxiliary information to a malicious user. This creates an opportunity to disclose the content of the corresponding plaintext by performing the aforementioned brute-force dictionary attack launched by the malicious CSP.
ii. A malicious user shares the file key in possession to the CSP, which could be utilized to decrypt the corresponding ciphertext uploaded by other users. The reason is that in order to achieve ciphertext deduplication, the key is

actually shared by more than one users. Consider the collision property of the deduplication index defined by the short hash value, it is even utilized to encrypt multiple files.

C. Design Goals

To enable efficient cloud storage with resistance against brute-force dictionary attack, side channel attack as well as the more sophisticated collusion attack, the design of our scheme should satisfy the following security and performance goals.

i. Data confidentiality: to ensure adversaries cannot derive the content of plaintext from the data stored in the cloud storage.
ii. Brute-force attack resistance: to ensure both the internal and external adversaries cannot infer the existence privacy for target data with low min-entropy by launching a brute-force dictionary attack.
iii. Side channel attack resistance: to ensure the external adversaries cannot infer the existence privacy of the target data in cloud storage according to the differentiated responses returned by the CSP during the process of deduplication.
iv. Collusion attack resistance: to ensure even if a more sophisticated collusion attack is launched by the malicious CSP and users, the confidentiality of the target data is still well protected.
v. Lightweight: to implement an only two-party involved system with minimum amount of communication overhead introduced. Meanwhile, the deduplication efficiency is ensured to be maximized.

IV. The Proposed Scheme

A. Notations

To construct a secure deduplication protocol, we assume DSDE consisting of the following algorithms is available:

i. $k \leftarrow KG(1^{\lambda})$: a key generation algorithm based on pseudo random number generator (PRNG), where a security parameter λ is introduced as input, and a symmetric key k is defined as output.
ii. $C \leftarrow E_{k_F}(F)$: a symmetric encryption algorithm for data encryption, where a random key k_F and a plaintext F are defined as input, and the ciphertext C is as output.
iii. $F \leftarrow D_{k_F}(C)$: a symmetric encryption algorithm for data decryption, where a random key k_F and a ciphertext C are defined as input, and the plaintext F is as output.
iv. $h \leftarrow H(F)$: a secure map-to-point cryptographic hash function, where a plaintext F is as input, and a hash value h is as output.
v. $hc \leftarrow \varepsilon_r(F)$: a partially homomorphic encryption algorithm for data encryption, which is defined to be consistent with that in [23]. In this algorithm, a random number r and a plaintext F are defined as input, and the ciphertext hc is as output.
vi. $fea \leftarrow FG(F)$: a feature generation algorithm, where a plaintext F and its feature fea are defined as input and output respectively. Specifically, a consistent feature should be ensured to be obtained from similar files with a high probability.
vii. $(pkg_1, pkg_2) \leftarrow PG(F, fea)$: a packages generation algorithm based on bloom filter, where the plaintext F and its feature fea are defined as input, and two packages pkg_1 and pkg_2 as output.

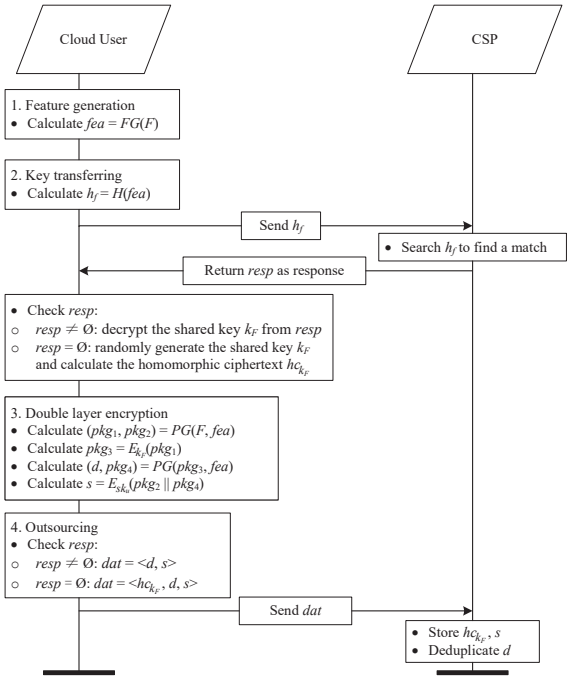


Fig. 2. Framework of DSDE.

B. Framework

In this part, we firstly present the framework of DSDE as a preparation before introducing the design in detail. According to our design, the data to be uploaded is split into two packages by employing a bloom filter, in which the larger one is encrypted by a randomly generated key. In order to achieve further deduplication, the key has to be transferred to subsequent owners, thus called the shared key. The remaining small package containing a small portion of the content in the aforementioned encrypted package is encrypted by the secret key of the user, which cannot be deduplicated at all. In this way, the offline brute-force attack and collusion attack can be well resisted since even though the shared key is vulnerable (see Section III-B for more detail), the corresponding package cannot be decrypted at all because the other part of it is protected by another key.

Specifically, we consider a file F to be outsourced by the cloud user $\mathcal{U}$. As shown in Fig. 2, $\mathcal{U}$ firstly computes the feature of F: $fea = FG(F)$. Secondly, an index $h_f = H(fea)$ is sent to the CSP to request a shared key k_F which is transferred among data owners by means of homomorphic encryption. For the initial uploader, since the index in request is not existent in the cloud, the CSP returns $resp = \emptyset$ as a response to

require the generation of the shared key k_F. In this case, k_F is homomorphically encrypted into hc_{k_F} before being utilized in subsequent key transferring. Otherwise, once $\mathcal{U}$ is not the initial one, it has to derive the shared key k_F from the response $resp$ returned by the CSP. Thirdly, F is split into pkg_1 and pkg_2 based on the result of the bloom filter, which takes fea as input. In particular, pkg_2 is formed by the bits extracted from the specified positions of F, and the remaining parts are aggregated together to be pkg_1. Next, through utilizing the shared key k_F, pkg_1 is encrypted into pkg_3 which is then split into two packages d and pkg_4 accordingly. After that, $pkg_2||pkg_4$ is encrypted by the secret key of the user sk_u into a secret package s. Finally, two packages d and s are sent to the CSP for storage, where d can be deduplicated. Specially, hc_{k_F} should be sent to the CSP as well if k_F is locally generated.

C. Construction of DSDE

1) Feature generation

According to the procedure introduced in Section IV-B, before data outsourcing, the cloud user $\mathcal{U}$ needs to generate a deduplication request index at first, which is utilized in subsequent shared key transferring and double layer encryption. Taking the offline brute-force attack in the scenario of index collision and the side channel attack in deduplication (see Section III-B for more detail) into account, the collisional property of deduplication request index should be supported. As a result, the feature of the file is extracted by employing the Rabin fingerprint algorithm [31], whose hash value is defined to be the index. Thanks to the design, there is a great probability to extract the same feature from similar files in fact, so that the attacks mentioned above are avoided. The specific process is divided into the following three steps.

i. For the outsourced file F, $N(N\in N^*)$ Rabin fingerprints can be returned over 64-byte sliding windows (each sliding window returns one Rabin fingerprint P_i ($i\in[1,N]$)).
ii. N pairs of coefficients (a_i, m_i) ($i\in[1,N]$ and $a_i, m_i\in N^*$) are used in N linear functions $\pi_i(x)$, where the i-th one is

$$\pi_i(x) = a_i \times x + m_i \mod 2^{64}. \quad (1)$$

Thus, N sub-features can be derived, in which the i-th one is $S_i = \pi_i(P_i)$.
iii. The feature $FG(F)$ is obtained by concatenating all N sub-features before calculating the cryptographic hash value. We have

$$FG(F) = H(S_1||\cdots||S_N). \quad (2)$$

Since $FG(F)$ is utilized to encrypt the shared key in the following process, it cannot be directly sent to the CSP for security reasons. Thus we define the deduplication request index to be $H(FG(F))$.

2) Shared key transferring

In order to achieve transferring of the shared keys in the way of two-party interactions among different data owners, the CSP has to keep the key related auxiliary information generated by the initial uploader, which is designed to be protected by $FG(F)$. In this way, the shared key can be transferred among owners of similar files. Furthermore, due to the high collisional property of $FG(F)$, both the offline brute-force attack in the scenario of index collision and the side channel attack in deduplication are able to be prevented. The process of shared key transferring is described in detail as follows.

i. $\mathcal{U}$ sends the uploading request index $H(FG(F))$ to the CSP.
ii. On receiving the index, the CSP checks its existence in the index table $\mathcal{I}$, and takes action according to the result.
   - $\mathcal{I}(H(FG(F))) = \emptyset$: the CSP will return an empty set $\emptyset$, which means that neither F nor its similar files have been stored in the cloud beforehand. In other words, there is no shared key for the requested file.
   - $\mathcal{I}(H(FG(F))) \neq \emptyset$: the CSP will return a matching record $\mathcal{M} = \langle \varepsilon_{r'}(k_{F'} \ominus FG(F^{'})), E_{\varepsilon_{r'}(k_{F'})}(E_{FG(F')}(k_{F'}))\rangle$, which means either the requested file or its similar version has been uploaded. Thus the corresponding key is existent and able to be transferred.
iii. The user performs the corresponding operations according to the response $resp$ received from the cloud. In particular, there are two cases of $resp$ as follows.
   - $resp = \emptyset$: $\mathcal{U}$ has to randomly generate a key $k_{random} = KG(1^{\lambda})$ locally.
   - $resp \neq \emptyset$: $\mathcal{U}$ cannot decrypt $k_{F'}$ from $\varepsilon_{r'}(k_{F'} \ominus FG(F^{'}))$ directly since r' is an unknown secret number generated by another user. Thus, $\mathcal{U}$ selects a random number r and calculates $\varepsilon_r(FG(F))$, which is utilized to derive

$$\varepsilon_{r'r}(k_{F'}) = \varepsilon_{r'}(k_{F'} \ominus FG(F^{'})) \oplus \varepsilon_r(FG(F)). \quad (3)$$

After that, $\mathcal{U}$ calculates $\varepsilon_{r'}(k_{F'})$ with the help of the inverse element r^{-1} and finally obtains

$$k_{F'} = D_{FG(F)}(D_{\varepsilon_{r'}(k_{F'})}(enc)), \quad (4)$$

where $enc = E_{\varepsilon_{r'}(k_{F'})}(E_{FG(F')}(k_{F'}))$. It should be noted that $FG(F) = FG(F^{'})$ in this case, because only when this condition is satisfied will the CSP return a matching record $\mathcal{M}$ as the response $resp$. The correctness of Equation (3) has been proved in [23].

Finally, $\mathcal{U}$ has $k_F = k_{random}$ or $k_F = k_{F'}$, either of which is utilized in the following encryption.

3) Double Layer Encryption

According to the analysis in Section III-B, a shared key is vulnerable to offline brute-force attack in the scenario of key acquisition, let alone the more sophisticated collusion attack. To solve above problems with data deduplication supported, we propose a double layer encryption mechanism. Specifically, the file to be outsourced is split into two packages, one of

which is encrypted by the shared key, and the other is by the secret key of the cloud user. Thus even though the shared key is obtained by an adversary, the first package cannot be decrypted directly at all since a part of its ciphertext is still protected by the secret key of the user. The process of double layer encryption is consisted of two core parts: packages generation and encryption.

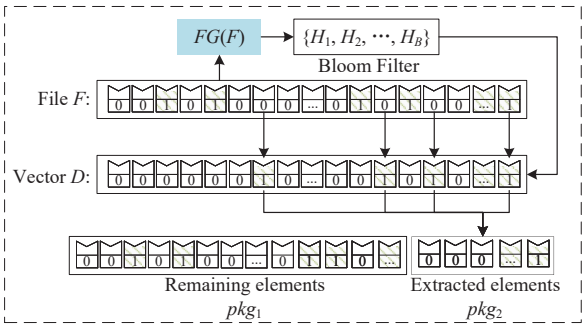


Fig. 3. The process of packages generation.

Packages Generation. In this part, we firstly elaborate on the detail of packages generation algorithm defined in Section IV-A with the bloom filter based position selection method. In particular, the feature generated in Section IV-C1 is defined to be the input of the bloom filter, which guarantees the bits from consistent positions are extracted from similar files with great probability to produce the package. Such a design is helpful in increasing the probability to produce the same packages from similar files. As shown in Fig. 3, the specific process consists of the following three steps.

i. We set the bloom filter which contains $B(B{\in}N^*)$ independent hash functions $\{H_1, H_2, \ldots, H_B\}$, in which $H_i(\cdot) = H(\cdot||i) \mod l_F$: $\{0,1\}^*{\rightarrow}[0, l_F{-}1]$ $(i \in [1, B]$, $l_F{\in}N^*)$. The output of each function is mapped to a specific position of a zero vector D with fixed l_F-bit length, which is consistent with the length of F.
ii. The feature $FG(F)$ is used as input to calculate hash values for each hash function and corresponding elements in the vector D are set to 1 successively regardless of its original value.
iii. Record the positions of non-zero elements in vector D, and extract data from the same positions in F to merge into package pkg_2, the length of which is rather small comparing with that of F. And the remaining parts are merged into package pkg_1.

Finally, two packages pkg_1 and pkg_2 are output as the result of function $PG(F, FG(F))$.

Packages Encryption. After splitting the file F into two packages pkg_1 and pkg_2, double layer encryption will be performed successively. The first layer with a shared key to achieve deduplication, and the second layer with a secret key to ensure security against brute-force and collusion attacks. As shown in Fig. 4, the detail of packages encryption is described as follows.

i. $\mathcal{U}$ encrypts the package pkg_1 with the shared key k_F to generate the ciphertext $pkg_3 = E_{k_F}(pkg_1)$.
ii. Split pkg_3 into two packages d and pkg_4 according to the same process of file splitting, in which pkg_4 is a small part of pkg_3, and d is consisted of the remaining elements. In particular, $FG(F)$ is defined to be the input of the bloom filter.
iii. A secret key $sk_u = KG(1^\lambda)$ is randomly generated by $\mathcal{U}$ to derive $s = E_{sk_u}(pkg_2||pkg_4)$.

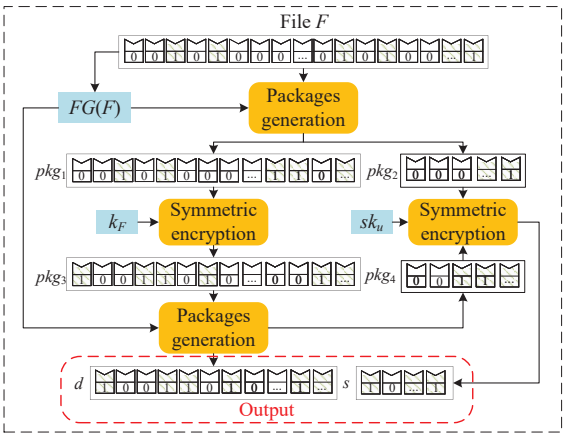


Fig. 4. The process of double layer encryption.

Finally, d and s are output as the ciphertexts.

4) Outsourcing and downloading

In this part, we introduce the processes of ciphertexts outsourcing and downloading. In particular, on finishing downloading, we focus on ciphertext decryption.

Outsourcing. During ciphertext outsourcing, the content of transmitted data is determined by the result of shared key transferring introduced in Section IV-C2, which can be classified into the following two cases:

- $k_F = k_{F'}$: $\mathcal{U}$ only needs to upload two packages $\langle d, s\rangle$ to the CSP.
- $k_F = k_{random}$: $\mathcal{U}$ derives $\varepsilon_r(k_F{\ominus}FG(F))$ and $E_{\varepsilon_r(k_F)}(E_{FG(F)}(k_F))$, both of which is utilized to transfer the key k_F to subsequent potential owners. Then, a tuple $\langle \varepsilon_r(k_F{\ominus}FG(F)), E_{\varepsilon_r(k_F)}(E_{FG(F)}(k_F)), d, s\rangle$ is uploaded to the CSP.

On receiving the ciphertexts, the CSP performs deduplication to d, and keeps s in storage to ensure security.

Decryption. When $\mathcal{U}$ requests to download the ciphertext of file F, the CSP will return a tuple $\langle \varepsilon_r(k_F{\ominus}FG(F)), E_{\varepsilon_r(k_F)}(E_{FG(F)}(k_F)), d, s\rangle$ as response. Then $\mathcal{U}$ can decrypt the ciphertexts according to the following steps.

i. $\mathcal{U}$ decrypts s with the secret key sk_u to generate $pkg_2||pkg_4 = D_{sk_u}(s)$.
ii. $\mathcal{U}$ obtains k_F from $\varepsilon_r(k_F{\ominus}FG(F))$ and $E_{\varepsilon_r(k_F)}(E_{FG(F)}(k_F))$ according to Equations (3) and (4) introduced in Section IV-C2.

iii. pkg_3 is recovered from two packages d and pkg_4 according to the inverse process of file splitting with $FG(F)$ as input of the bloom filter. Then pkg_1 can be derived as $pkg_1 = D_{k_F}(pkg_3)$.

iv. F is recovered from packages pkg_1 and pkg_2 according to the same procedure to recover pkg_3.

V. Security Analysis

***Lemma* 1.** *For a certain predictable file with low-min entropy, the corresponding deduplication request index can be duplicated among its possible versions with a great probability.*

Proof: Suppose F is a predictable file with low-min entropy, whose n possible versions are denoted by $F_1, F_2, \ldots, F_n$ respectively. Consider F_i and F_j $(i, j \in [1, n], i \neq j)$, denote the total number of identical bytes to be $b_{i,j}$ $(0 < b_{i,j} < \min\{l_{F_i}, l_{F_j}\})$ in which l_{F_i} and l_{F_j} are their lengths respectively. It is worth mentioning that as long as the order of common bytes is consistent, their length is considered, regardless of whether they are consecutive or not.

Since the result of $FG(F)$ is generated according to the first $64 + N - 1$ bytes of F (see Section IV-C1 for more detail), the same deduplication request index can be obtained from F_i and F_j as long as $Head(F_i, 63 + N) = Head(F_j, 63 + N)$, in which the function $Head(F, len)$ is defined to extract the first len bytes of F. Thus, the probability that the same deduplication request index is derived from F_i and F_j in the condition of $b_{i,j} < 63 + N$ can be formulated as in Equation (5).

$$\Pr[FG(F_i) = FG(F_j) | b_{i,j} < 63 + N] = 0. \tag{5}$$

On the other hand, if $b_{i,j} \geq 63 + N$, the above probability can be formulated as in Equation (6).

$$\begin{aligned} &\Pr[FG(F_i) = FG(F_j) | b_{i,j} \geq 63 + N] \\ =&\frac{C_{l_{F_i}-(63+N)}^{b_{i,j}-(63+N)} \times C_{l_{F_j}-(63+N)}^{b_{i,j}-(63+N)}}{C_{l_{F_i}}^{b_{i,j}} \times C_{l_{F_j}}^{b_{i,j}}}. \end{aligned} \tag{6}$$

Taking $C_{l_{F_i}-(63+N)}^{b_{i,j}-(63+N)} / C_{l_{F_i}}^{b_{i,j}}$ for example, it can be simplified as follows.

$$C_{l_{F_i}-(63+N)}^{b_{i,j}-(63+N)} / C_{l_{F_i}}^{b_{i,j}} = \prod_{k=b_{i,j}+1}^{l_{F_i}} (1 - \frac{63 + N}{k}). \tag{7}$$

Similarly, we can infer that $C_{l_{F_j}-(63+N)}^{b_{i,j}-(63+N)} / C_{l_{F_j}}^{b_{i,j}} = \prod_{k=b_{i,j}+1}^{l_{F_j}} (1 - \frac{63+N}{k})$. Thus, Equation (6) can be simplified to Equation (8).

$$\begin{aligned} &\Pr[FG(F_i) = FG(F_j) | b_{i,j} \geq 63 + N] \\ =&\prod_{k=b_{i,j}+1}^{l_{F_i}} (1 - \frac{63 + N}{k}) \times \prod_{k=b_{i,j}+1}^{l_{F_j}} (1 - \frac{63 + N}{k}). \end{aligned} \tag{8}$$

Since $F_1, F_2, \ldots, F_n$ are possible versions of the low-min entropy file F, because of their high similarity, for any F_i and F_j $(i, j \in [1, n], i \neq j)$, $b_{i,j}$ approaches $l_{F_i} \approx l_{F_j} \gg 63 + N$. As a result, $1 - (63 + N)/k$ approaches 1 for each $k \in [b_{i,j} + 1, l_{F_i}]$ or $k \in [b_{i,j} + 1, l_{F_j}]$. Thus, $\Pr[FG(F_i) = FG(F_j)]$ is equal to the result of Equation (8) which approaches 1. Therefore, we can draw the conclusion that for a certain low-min entropy file, the corresponding deduplication request index can be duplicated among its possible versions with a great probability. ■

***Theorem* 1.** *In a certain deduplication checking, DSDE is able to achieve protection of existence privacy under the side channel attack.*

Proof: Suppose that F is the target file whose existence privacy in cloud storage is of an attacker's interest. Consider the adversary generates n possible versions of F by launching a brute-force attack, each of which is denoted by $F_1, F_2, \ldots, F_n$ respectively. For each F_i $(i \in [1, n])$, the adversary calculates an index $H(FG(F_i))$ as the deduplication request before outsourcing to the CSP.

In particular, if the index is non-duplicate, the probability that the ciphertext of F_i is existent in cloud storage can be formulated as $\Pr[\delta_i \in S | \bar{A}_i] = 0$, where $\delta_i \in S$ denotes the ciphertext of F_i is existent in the cloud, and $\bar{A}_i$ means the event that the index of F_i is absent. Thus, we have another conclusion that $\Pr[\delta_i \notin S | \bar{A}_i] = 1$, which means that only the inexistence privacy of δ_i is exposed. In this case, a random key needs to be generated.

On the other hand, if $H(FG(F_i))$ is duplicate, the adversary actually cannot simply determine whether the target file is existent because of its high collision rate. According to the design of DSDE, we can simply draw the conclusion that A_i is equivalent to the event that the generation of a new random key for F_i is not needed. Thus if the generation of a new random key is not needed, the probability that δ_i is existent can be formulated as

$$\begin{aligned} &\Pr[\delta_i \in S | A_i] \\ =&\frac{\Pr[A_i | \delta_i \in S] \times \Pr[\delta_i \in S]}{\Pr[A_i | \delta_i \notin S] \times \Pr[\delta_i \notin S] + \Pr[A_i | \delta_i \in S] \times \Pr[\delta_i \in S]} \end{aligned} \tag{9}$$

It should be noted that $\Pr[A_i | \delta_i \in S] = 1$ apparently so that Equation (9) can be simplified to $\frac{\Pr[\delta_i \in S]}{\Pr[A_i | \delta_i \notin S] \times \Pr[\delta_i \notin S] + \Pr[\delta_i \in S]}$. Denote the value of $\Pr[\delta_i \in S]$ as x_i, which approaches 0 since the probability that an arbitrary file is in cloud is rather small. Then denote the value of $\Pr[A_i | \delta_i \notin S]$ as p_i. Thus the above result is equivalent to $\frac{x_i}{p_i(1-x_i)+x_i}$, which can be further reduced to $\frac{1}{p_i/x_i - p_i + 1}$. According to *Lemma* 1, $p_i > 0$. Thus $p_i \gg x_i$, and $0 < \Pr[\delta_i \in S | A_i] \ll 1$ can be obtained. It can also be inferred that $0 < \Pr[\delta_i \notin S | A_i] < 1$. Therefore, even if the event A_i occurs, the malicious user still cannot determine the existence of δ_i. As a result, the existence privacy of F_i is well protected by DSDE even though a side channel attack is launched. ■

***Lemma* 2.** *For a certain low-min entropy file, the corresponding package d can be deduplicated among its possible versions with a great probability.*

Proof: Suppose F is a low-min entropy file, whose n possible versions can be denoted by $F_1, F_2, \ldots, F_n$ respectively. Similar to *Lemma* 1, suppose F_i and F_j have $b'_{i,j}$ ($0 < b'_{i,j} < 8\times\min\{l_{F_i}, l_{F_j}\}, i,j \in [1,n], i\neq j$) identical bits with a consistent order, in which l_{F_i} and l_{F_j} are defined to be the length of F_i and F_j respectively. In order to ensure $d_i = d_j$, each of which is defined to be the first output of the double layer encryption (see Section IV-C3 for more detail) for F_i and F_j respectively, the following two conditions must be satisfied.

i. $FG(F_i) = FG(F_j)$: to ensure the same encryption key $k_{F_i} = k_{F_j}$ can be shared.
ii. $pkg_{i,1} = pkg_{j,1}$: to ensure the ciphertext $pkg_{i,3}$ is consistent with $pkg_{j,3}$, from each of which, d_i and d_j are derived respectively. It should be noted that $pkg_{i,1}$ and $pkg_{j,1}$ are the first output from the process of packages generation for F_i and F_j. $pkg_{i,3}$ and $pkg_{j,3}$ are their respective ciphertext encrypted by the shared key.

Then, we define Y_1, Y_2 and Y_3 to be the event of $FG(F_i) = FG(F_j)$, $b'_{i,j} \geq \max\{8\times l_{F_i} - B, 8\times l_{F_j} - B\}$ and $l_{F_i} = l_{F_j}$ respectively, where B (defined in Section IV-C3) is the bit length of the second output from the process of packages generation. If $\bar{Y_2}$ occurs, the number of different bits from F_i and F_j is greater than the value of B. Once B bits are extracted from F_i and F_j, the remaining packages $pkg_{i,1}$ and $pkg_{j,1}$ must be different. Similarly, if $\bar{Y_3}$ occurs, we have $l_{F_i}\neq l_{F_j}$, so that $pkg_{i,1}\neq pkg_{j,1}$ as a result. Thus, the following equations can be obtained.

$$\begin{cases} \Pr[d_i = d_j|\bar{Y_1}] = 0 \\ \Pr[d_i = d_j|\bar{Y_2}] = 0 \\ \Pr[d_i = d_j|\bar{Y_3}] = 0 \end{cases}. \tag{10}$$

According to Equation (10), the probability that $d_i = d_j$ can be formulated as in the Equation (11).

$$\begin{aligned} \Pr[d_i = d_j] =& \Pr[d_i = d_j|Y_1Y_2Y_3] \\ &\times \Pr[Y_1]\times \Pr[Y_2]\times \Pr[Y_3]. \end{aligned} \tag{11}$$

Then, in order to ensure $pkg_{i,1} = pkg_{j,1}$, every one of the bits in packages $pkg_{i,1}$ and $pkg_{j,1}$ must be from $b'_{i,j}$ during the process of package generation. Thus, the value of $\Pr[d_i = d_j|Y_1Y_2Y_3]$ can be calculated in Equation (12) as follows.

$$\begin{aligned} \Pr[d_i = d_j|Y_1Y_2Y_3] =& \frac{C_{b'_{i,j}}^{8l_{F_i}-B}}{C_{8l_{F_i}}^{8l_{F_i}-B}} \\ =& \prod_{k=B+1}^{8l_{F_i}} (1 - \frac{8l_{F_i} - b'_{i,j}}{k}). \end{aligned} \tag{12}$$

Since $F_1, F_2, \ldots, F_n$ are possible versions of the low-min entropy file F, their content is highly similar. Thus $b'_{i,j}$ approaches $8l_{F_i} = 8l_{F_j}$, so that $1-(8l_{F_i} - b'_{i,j})/k$ approaches 1 for each $k \in [B+1, 8l_{F_i}]$ or $k \in [B+1, 8l_{F_j}]$. So the result of Equation (12) approaches 1, $\Pr[Y_1]$ approaches 1 (proved in *Lemma* 1), and $Pr[Y_2]$, $Pr[Y_3]\gg 0$ for the possible versions F_i, F_j. Finally, we have $\Pr[d_i = d_j]\gg 0$. As a result, we can draw the conclusion that for a certain low-min entropy file, the package d corresponds to this file can be duplicated among its possible versions with a great probability.

■

***Theorem* 2.** *For a certain low-min entropy file, DSDE is able to achieve protection of confidentiality under the offline brute-force attack.*

Proof: Suppose that F is the target file whose content is of an attacker's interest. Consider the adversary generates n possible versions of F by launching a brute-force attack, each of which is denoted by $F_1, F_2, \ldots, F_n$ respectively. For each F_i ($i \in [1,n]$), the malicious CSP calculates $H(FG(F_i))$ as the deduplication request index. Consider two scenarios to launch such an attack introduced in our threat model (see Section III-B for more details).

Index collision: As proved in *Theorem* 1, if the index is non-duplicate, the CSP is only able to know that the ciphertext of F_i is not existent. Meanwhile, $\Pr[\delta_i\in S|A_i]\ll 1$, which means that whether the ciphertext of F_i is existent or not cannot be simply inferred from the existence status of the deduplication request index. Thus the confidentiality of the target file is well protected by DSDE in this scenario even though the deduplication request index $H(FG(F_i))$ for F_i is obtained by the malicious CSP through offline brute-force attack.

Key acquisition: According to the analysis in Section III-B, the shared key utilized in the generation of package d is vulnerable to offline brute-force attack. Thus, for each F_i ($i \in [1,n]$), the adversary can calculate a package d_i as the first output of the double layer encryption. Similar to *Theorem* 1, if d_i is non-duplicate, the probability that F_i has been outsourced by a user after encryption can be formulated as

$$\begin{cases} \Pr[W_i|\bar{Z_i}] = 0 \\ \Pr[\bar{W_i}|\bar{Z_i}] = 1 \end{cases}, \tag{13}$$

where W_i denotes the event that F_i has been outsourced to the CSP, and $\bar{Z_i}$ represents the event that d_i is non-duplicate. In this case, only incorrectness of the version F_i could be inferred.

On the other hand, if d_i is duplicate, the probability that F_i has been outsourced can be formulated as

$$\Pr[W_i|Z_i] = \frac{\Pr[Z_i|W_i]\times \Pr[W_i]}{\Pr[Z_i|\bar{W_i}]\times \Pr[\bar{W_i}] + \Pr[Z_i|W_i]\times \Pr[W_i]}. \tag{14}$$

It should be noted that $\Pr[Z_i|W_i] = 1$ apparently so that Equation (14) can be simplified as $\frac{\Pr[W_i]}{\Pr[Z_i|\bar{W_i}]\times \Pr[\bar{W_i}]+\Pr[W_i]}$. Denote the value of $\Pr[W_i]$ as x'_i, which approaches 0 since the probability that an arbitrary file has been outsourced is rather small, and the value of $\Pr[Z_i|\bar{W_i}]$ as q_i. The above result is equivalent to $\frac{x'_i}{q_i(1-x'_i)+x'_i}$, which can be further reduced to $\frac{1}{q_i/x'_i - q_i + 1}$. According to *Lemma* 2, $\Pr[Z_i|\bar{W_i}] > 0$. Thus the

value of q_i is much greater than $x_i^{'}$, and the result of Equation (14) < 1. Therefore, the malicious CSP cannot confirm the correctness of the version F_i in this scenario even though d_i is existent in the cloud. As a result, DSDE is able to achieve protection of confidentiality under the offline brute-force attack.

■

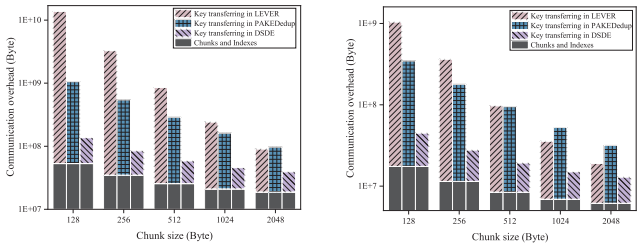
VI. Performance Evaluation

In this section, we take experiments on two real world datasets: Enron Email [32] and Linux Logs [33], to evaluate the efficiency of DSDE. Specifically, we select 2,000 files with sizes of 20 KB to 30 KB from the first dataset, and 500 files with sizes of 20 KB to 30 KB from the second one. To achieve chunk-level deduplication, we divide each file into chunks of fixed size, and employ padding strategy to make sure the length of the last chunk is consistent with the other ones. Furthermore, DSDE is compared with the other two state-of-the art non-deterministic encryption based deduplication schemes: LEVER [23] and PAKEDedup [28], in terms of communication and storage overhead. The processes of CSP are implemented on Amazon elastic computing cloud (EC2) instances, and client-side ones on a workstation with an Apple M2 CPU @ 3.5 GHz, 16 GB RAM and a 512 GB solid state drive. All algorithms are implemented using Python version 3.6.8. In particular, the SEAL library version 3.3 is employed for homomorphic encryption, the PyCryptodome library version 3.17 is for symmetric encryption, and the mmh3 library version 3.1.0 is for the hash functions in Bloom Filter. It should be noted that all presented results are obtained as the average of 20 independent trials.

A. Communication Overhead

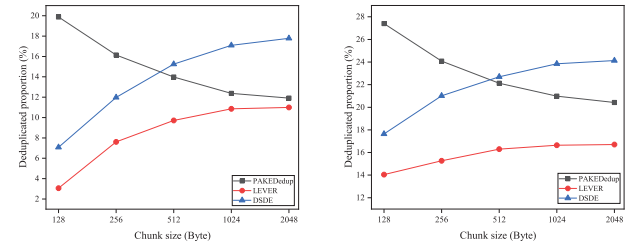
In this part, we compare DSDE with the other two state-of-the art schemes in terms of the client-side communication cost under different chunk size. At first, we fix the length of the short hash value in LEVER and PAKEDedup to be 1 B and the persistence probability of a certain chunk to 0.3, which is consistent with the setting in [23]. Moreover, the number of checkers in PAKEDedup is set to be 30, which is the same with the setting in [28]. In order to achieve the pre-determined persistence probability, we randomly select about 30% of files from each one of the datasets to upload to the cloud beforehand. Then every one of the files are requested to be deduplicated for performance evaluation. In particular, to achieve chunk-level deduplication, test files selected from both datasets are divided into chunks of a certain size, ranging from 128 B to 1,024 B. In addition, we fix the length of the key in symmetric and asymmetric encryption to be 32 B and 128 B respectively. The communication overhead under different chunk size is compared and the results are shown in Fig. 5.

It is clear from the results that DSDE is more efficient than LEVER and PAKEDedup in communication overhead. Take Enron Email dataset with chunk size of 1,024 B for example, the communication overhead for LEVER and PAKEDedup are 2.22×10^8 and 1.40×10^8 B, each of which is much greater

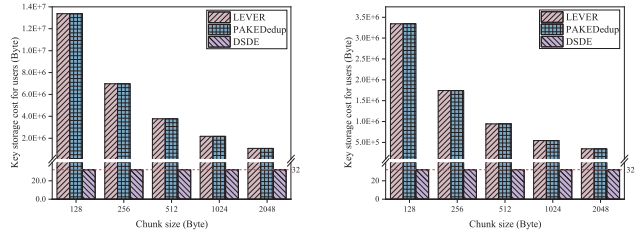


(a) Enron Email dataset (b) Linux Logs dataset

Fig. 5. Comparison of the communication overhead under different chunk size.



(a) Cloud storage overhead in Enron Email dataset (b) Cloud storage overhead in Linux Logs dataset



(c) Key storage overhead for users in Enron Email dataset (d) Key storage overhead for users in Linux Logs dataset

Fig. 6. Comparison of the storage overhead under different chunk size.

than 2.48×10^7 B for DSDE. The reason is that a certain short hash value in LEVER may correspond to multiple random keys, each of which has to be utilized to encrypt a single chunk before outsourcing the corresponding ciphertext. As a comparison, because of the one-to-one correspondence between the key and the deduplication request index in DSDE, redundant uploading is avoided. While for PAKEDedup, it is low in efficiency because of the same-input-PAKE protocol employed for key transferring among checkers. Furthermore, among the three schemes in comparison, the communication overhead decreases with the increase of chunk size. The reason is that for data of a certain size, the number of chunks decreases with the increase of chunk size, which restrains the number of indexes and keys in turn.

B. Storage Overhead

In this part, we compare the storage overhead of DSDE for the CSP and the cloud user respectively with the other two state-of-the art schemes under different chunk size. In particular, the cloud storage overhead is evaluated in terms of the deduplicated proportion, and the user side overhead is in terms of the key storage overhead. All settings in this

experiment are consistent with those in the previous part, and the results are shown in Fig. 6 respectively.

According to Fig. 6 (a) and (b), it is clear that DSDE is more efficient in cloud storage overhead than the other two state-of-the arts when the size of chunks is large enough. Take Enron Email dataset for example, when the chunk size is 1,024 B, the deduplicated proportion for LEVER and PAKEDedup are 10.86% and 12.38% respectively, each of which is smaller than 17.93% in DSDE. The main reason is that a consistent package d can be generated from similar chunks with a great probability in DSDE, which in turn compensates for the slight performance degradation caused by package s. Moreover, because of the comparative small number of chunks, the corresponding keys are also restrained, which is reflected in the low storage overhead for keys. However, as shown in the first two figures, when the size of chunks is not large enough, the cloud storage overhead of DSDE is greater than that of PAKEDedup. The reason is that homomorphically encrypted random keys should be stored in the cloud for both LEVER and DSDE, the number of which grows rapidly with that of chunks. As a comparison, such an overhead is not required in PAKEDedup since random keys are stored locally by users. As a result, a heavy storage burden is introduced to users, which is coincide with that of LEVER. The corresponding results are shown in Fig. 6 (c) and (d) respectively. It is worth mentioning that on the side of users, only secret keys of users need to be stored.

VII. Conclusion

We propose DSDE as a novel cross-user ciphertext deduplication scheme to deal with security problems such as brute-force attacks and the collusion attack in a lightweight way. With the help of the proposed data splitting based double layer encryption, the aforementioned attacks can be well resisted. Moreover, because of the high collisional property of the deduplication request index utilized, the associated key can be shared. According to our design, the same part of similar data can be deduplicated after encryption with a great probability as a result, which improves the performance of deduplication. The security analysis and experimental results show that DSDE is able to protect data confidentiality during the process of ciphertext deduplication with performance guaranteed comparing with the state-of-the art.

Acknowledgment

This work was specially supported by National Natural Science Foundation of China (62102113), Fundamental Research Funds for the Central Universities, University of International Relations (3262023T30). We would like to thank the anonymous reviewers for their careful reading, helpful comments, and constructive suggestions, which is of great importance to improve the quality of our manuscript.

References

[1] IDC, "The Digitization of the World: From Edge to Core," *https://www.seagate.com/files/www-content/our-story/trends/files/idc-seagate-dataage-whitepaper.pdf.*

[2] J. Liang, Z. Qin, S. Xiao, L. Ou, and X. Lin, "Efficient and Secure Decision Tree Classification for Cloud-assisted Online Diagnosis Services," *IEEE Transactions on Dependable and Secure Computing*, vol. 18, no. 4, pp. 1632–1644, 2021.

[3] K. Xue, N. Gai, J. Hong, D. Wei, P. Hong and N. Yu, "Efficient and Secure Attribute-based Access Control with Identical Sub-Policies Frequently Used in Cloud Storage," *IEEE Transactions on Dependable and Secure Computing*, vol. 19, no. 1, pp. 635-646, 2022.

[4] M. Li, L. Zhu and X. Lin, "Privacy-Preserving Traffic Monitoring with False Report Filtering via Fog-Assisted Vehicular Crowdsensing," *IEEE Transactions on Services Computing*, vol. 14, no. 6, pp. 1902-1913, 2021.

[5] P. Zuo, Y. Hua, C. Wang, W. Xia, S. Cao, Y. Zhou and Y. Sun, "Mitigating Traffic-Based Side Channel Attacks in Bandwidth-Efficient Cloud Storage," in *Proceedings of the 32nd IEEE International Parallel and Distributed Processing Symposium* (*IPDPS*), Vancouver, BC, Canada, pp. 1153-1162, 2018.

[6] C. Yu, S. P. Gochhayat, M. Conti and C. Lu, "Privacy Aware Data Deduplication for Side Channel in Cloud Storage," *IEEE Transactions on Cloud Computing*, vol. 8, no. 2, pp. 597-609, 2020.

[7] X. Tang, Y. Zhu and M. Fu, "Comments on 'Privacy Aware Data Deduplication for Side Channel in Cloud Storage'," *IEEE Transactions on Cloud Computing*, doi: 10.1109/TCC.2024.3376996.

[8] R. Vestergaard, Q. Zhang and D. E. Lucani, "CIDER: A Low Overhead Approach to Privacy Aware Client-side Deduplication," in *Proceeding of the 33rd IEEE Global Communications Conference* (*GLOBECOM*), Taipei, Taiwan, pp. 1-6, 2020.

[9] X. Tang, X. Chen, R. Zhou, L. Sui and T. Zhou, "Marking based Obfuscation Strategy to resist Side Channel Attack in Cross-User Deduplication for Cloud Storage," in *Proceedings of the 21st IEEE International Conference on Trust, Security and Privacy in Computing and Communications* (*TrustCom*), Wuhan, China, pp. 547-555, 2022.

[10] X. Tang, Z. Liu, Y. Shao and H. Di, "Side Channel Attack resistant Cross-User Generalized Deduplication for Cloud Storage," in *Proceedings of the 35th IEEE International Conference on Communications* (*ICC*), Seoul, Republic of Korea, pp. 998-1003, 2022.

[11] G. Ha, H. Chen, C. Jia and M. Li, "Threat Model and Defense Scheme for Side-Channel Attacks in Client-Side Deduplication," *Tsinghua Science and Technology*, vol. 28, no. 1, pp. 1-12, 2023.

[12] J. R. Douceur, A. Adya, W. J. Bolosky, P. Simon and M. Theimer, "Reclaiming Space from Duplicate Files in a Serverless Distributed File System," in *Proceedings of the 22nd International Conference on Distributed Computing Systems* (*ICDCS*), Vienna, Austria, pp. 617-624, 2002.

[13] M. Bellare, S. Keelveedhi and T. Ristenpart, "Message-Locked Encryption and Secure Deduplication," in *Proceedings of the 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques* (*Eurocrypt*), Athens, Greece: Springer, pp. 296–312, 2013.

[14] M. Bellare, S. Keelveedhi and T. Ristenpart, "DupLESS: Server-aided Encryption for Deduplicated Storage," in *Proceedings of the 22nd USENIX Security Symposium* (*USENIX Security*), Berkeley, CA, USA, pp. 179-194, 2013.

[15] Q. Jia, G. Ha, H. Wang, H. Ma and H. Chen, "An Enhanced MinHash Encryption Scheme for Encrypted Deduplication," in *Proceedings of the 21st IEEE International Conference on Trust, Security and Privacy in Computing and Communications* (*TrustCom*), Wuhan, China, pp. 565-573, 2022.

[16] J. Li, Y. Ren, P. P. C. Lee, Y. Wang, T. Chen and X. Zhang, "FeatureSpy: Detecting Learning-Content Attacks via Feature Inspection in Secure Deduplicated Storage," in *Proceedings of the 18th IEEE Conference on Computer Communications* (*INFOCOM*), New York, NY, USA, pp. 1-10, 2023.

[17] S. Li, C. Xu, Y. Zhang, Y. Du and K. Chen, "Blockchain-based Transparent Integrity Auditing and Encrypted Deduplication for Cloud Storage," *IEEE Transactions on Services Computing*, vol. 16, no. 1, pp. 134-146, 2023.

[18] Y. Zhang, Y. Mao, M. Xu, F. Xu and S. Zhong, "Towards Thwarting Template Side-Channel Attacks in Secure Cloud Deduplications," *IEEE Transactions on Dependable and Secure Computing*, vol. 18, no. 3, pp. 1008-1018, 2021.

[19] Y. Duan, "Distributed Key Generation for Encrypted Deduplication Achieving the Strongest Privacy," in *Proceedings of the 6th edition of the ACM Workshop on Cloud Computing Security* (*CCSW*), Scottsdale, Arizona, USA, pp. 57-68, 2014.

[20] 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," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 9, pp. 4101–4112, 2018.
[21] Y. Zhang, C. Xu, N. Cheng and X. Shen, "Secure Password-Protected Encryption Key for Deduplicated Cloud Storage Systems," *IEEE Transactions on Dependable and Secure Computing*, vol. 19, no. 4, pp. 2789-2806, 2022.
[22] T. Jiang, X. Yuan, Y. Chen, K. Cheng, L. Wang, X. Chen and J. Ma, "FuzzyDedup: Secure Fuzzy Deduplication for Cloud Storage," *IEEE Transactions on Dependable and Secure Computing*, vol. 20, no. 3, pp. 2466-2483, 2023.
[23] Z. Pooranian, M. Shojafar, S. Garg, R. Taheri and R. Tafazolli, "LEVER: Secure Deduplicated Cloud Storage with Encrypted Two-Party Interactions in Cyber-Physical Systems," *IEEE Transactions on Industrial Informatics*, vol. 17, no. 8, pp. 5759-5768, 2021.
[24] K. Zhang, X. Liang, R. Lu and X. Shen, "Sybil Attacks and Their Defenses in the Internet of Things," *IEEE Internet of Things Journal*, vol. 1, no. 5, pp. 372-383, 2014.
[25] J. Bai, J. Yu and X. Gao, "Secure Auditing and Deduplication for Encrypted Cloud Data Supporting Ownership Modification," *Soft Computing*, vol. 24, no. 16, pp. 12197-12214, 2020.
[26] L. Liu, Y. Zhang and X. Li, "KeyD: Secure Key-Deduplication with Identity-based Broadcast Encryption," *IEEE Transactions on Cloud Computing*, vol. 9, no. 2, pp. 670-681, 2021.
[27] H. Yuan, X. Chen, J. Li, T. Jiang, J. Wang and R. H. Deng, "Secure Cloud Data Deduplication with Efficient Re-Encryption," *IEEE Transactions on Services Computing*, vol. 15, no. 1, pp. 442-456, 2022.
[28] J. Liu, N. Asokan and B. Pinkas, "Secure Deduplication of Encrypted Data without Additional Independent Servers," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security* (*CCS*), New York, NY, USA, pp. 874–885, 2015.
[29] A. F. Webster and Stafford E. Tavares, "On the Design of S-boxes," in *Proceedings of the 5th Annual International Cryptology Conference* (*CRYPTO*), Santa Barbara, CA, USA, pp. 523-534, 1985.
[30] X. Xiao, Z. Tang, C. Li, B. Xiao and K. Li, "SCA: Sybil-Based Collusion Attacks of IIoT Data Poisoning in Federated Learning," *IEEE Transactions on Industrial Informatics*, vol. 19, no. 3, pp. 2608-2618, 2023.
[31] M. O. Rabin, "Fingerprint by Random Polynomials," Center for Research in Computing Technology, Aiken Computation Laboratory, Harvard University, 1981.
[32] W. W. Cohen, "Enron email dataset," *https://www.cs.cmu.edu/./enron/*.
[33] Srinidhi, "LogPAI::Linux Logs dataset," *https://www.kaggle.com/datasets/ggsri123/linux-logs*.