Hindawi
Security and Communication Networks
Volume 2020, Article ID 3046595, 15 pages
https://doi.org/10.1155/2020/3046595

Research Article

Toward Serverless and Efficient Encrypted Deduplication in Mobile Cloud Computing Environments

Youngjoo Shin,[1] Junbeom Hur,[2] Dongyoung Koo,[3] and Joobeom Yun[4]

[1]*School of Computer and Information Engineering, Kwangwoon University, Seoul, Republic of Korea*
[2]*Department of Computer Science and Engineering, Korea University, Seoul, Republic of Korea*
[3]*Department of Electronics and Information Engineering, Hansung University, Seoul, Republic of Korea*
[4]*Department of Computer and Information Security, Sejong University, Seoul, Republic of Korea*

Correspondence should be addressed to Dongyoung Koo; dykoo@hansung.ac.kr and Joobeom Yun; jbyun@sejong.ac.kr

Received 16 February 2020; Revised 26 June 2020; Accepted 23 July 2020; Published 28 August 2020

Academic Editor: Luigi Coppolino

With the proliferation of new mobile devices, mobile cloud computing technology has emerged to provide rich computing and storage functions for mobile users. The explosive growth of mobile data has led to an increased demand for solutions that conserve storage resources. Data deduplication is a promising technique that eliminates data redundancy for storage. For mobile cloud storage services, enabling the deduplication of encrypted data is of vital importance to reduce costs and preserve data confidentiality. However, recently proposed solutions for encrypted deduplication lack the desired level of security and efficiency. In this paper, we propose a novel scheme for serverless efficient encrypted deduplication (SEED) in mobile cloud computing environments. Without the aid of additional servers, SEED ensures confidentiality, data integrity, and collusion resistance for outsourced data. The absence of dedicated servers increases the effectiveness of SEED for mobile cloud storage services, in which user mobility is essential. In addition, noninteractive file encryption with the support of lazy encryption greatly reduces latency in the file-upload process. The proposed indexing structure (D-tree) supports the deduplication algorithm and thus makes SEED much more efficient and scalable. Security and performance analyses prove the efficiency and effectiveness of SEED for mobile cloud storage services.

1. Introduction

Most mobile devices, such as smartphones and Internet of things products, are constantly connected to the Internet thanks to advances in mobile wireless network technology. Mobile cloud computing (MCC) [1, 2], also referred to as mobile edge computing [3], has emerged to fulfill the need for ubiquitous, low-latency services and applications for mobile users. Through the combination of cloud computing, mobile computing, and wireless networks, MCC provides a rich array of computing and storage options for mobile users [4].

With the explosive growth in the volume of data outsourced from mobile devices, it is crucial for mobile cloud service providers (MCSPs) to minimize the costs of storing outsourced data. Data deduplication, a technique that eliminates data redundancy, can achieve this goal and reduce resource use, including disk space and network bandwidth, by more than 90% [5].

To maintain confidentiality of the outsourced data, it is essential to devise a technique to conduct deduplication over encrypted data. As a first attempt for encrypted deduplication, convergent encryption (CE) [6, 7] was proposed. CE computes an encryption key from the hash of the data, thus generating identical ciphertexts from identical plaintexts. Although the method is quite simple, it is vulnerable to brute-force attacks [8, 9] because encryption keys are deterministically computed from plaintext, which makes them predictable. For example, given CE ciphertext C (of plaintext F) and a dictionary of possible plaintexts $D = \{F_1, F_2, \ldots, F_n\}$, an adversary might attempt to derive an encryption key for each plaintext in D and then perform encryption on it until C is found.

Server-aided encryption [8, 10–12] addresses this problem and aims to mitigate brute-force attacks on encrypted deduplication. This approach uses a dedicated key server for the generation of encryption keys. The key server possesses its own secret key and performs an oblivious key generation protocol [13] with users: for each request, it generates an encryption key, using the secret key and a blinded hash computed from the data, and then returns it to the user. By doing so, the randomness of the key server's secret key contributes to the encryption keys, which makes brute-force attacks infeasible while the secret key is kept hidden from adversaries.

Despite its resistance to brute-force attacks, server-aided encryption has several limitations when applied to an MCC environment. First, the achievement of security comes at the cost of managing key servers, which are subject to single point-of-failure or server-compromise attacks [8]. Second, the dedicated key servers that are usually residing within on-premises networks severely reduce user mobility, which significantly degrades the effectiveness and performance of the MCC technology. This restriction on mobility could be relieved by deploying multiple key servers over geographically separated areas. However, this not only incurs high deployment costs but also exposes the system to a wide variety of security threats.

For the successful provisioning of ubiquitous, low-latency, and secure storage services in a mobile cloud environment, it is necessary to devise *serverless encryption* that enables brute-force-resistant encrypted deduplication without the aid of additional servers.

In this paper, we propose a novel scheme for serverless and efficient encrypted deduplication (SEED) in MCC environments. Instead of using key servers, users perform bilinear pairing-based encryption on files using their own public and secret keys. The bilinearity of file encryption allows an equality test to be conducted for the ciphertexts generated under different secret keys and thus enables cross-user deduplication of encrypted data. The encryption algorithm randomizes all ciphertexts and the corresponding tags, which are susceptible to exposure to adversaries, using a random source supplied from the users' secret keys. The provable security ensures that no information about the plaintext is revealed from either ciphertexts or tags. Furthermore, file encryption allows the tags to be computed independently of the ciphertexts, which makes it possible for the ciphertexts and tags to be generated in parallel. This property of SEED enables *lazy encryption*, a novel feature in which ciphertext generation, a computationally expensive component of encryption, can be delayed or even omitted in the case of client-side deduplication.

In addition to bilinear pairing-based file encryption, SEED is based on an efficient deduplication algorithm. For this, we propose *D-tree*, a new indexing data structure that supports deduplication. D-tree is a random binary tree, which is a binary search tree that is formed from the random permutation of nodes. Each node in a D-tree contains a tag for an outsourced file as deduplication information within the storage. The cloud server (i.e., MCSP) can perform a binary search over the D-tree for identical files within the storage by running equality tests on each node. Because nodes are balanced in a random binary tree, D-tree preserves logarithmic computational complexity in the worst case for the deduplication algorithm.

SEED is significantly more practical for mobile cloud storage services than existing solutions because of the following advantages:

(i) It eliminates the need for key servers, which severely restricts user mobility. The absence of key servers also allows noninteractive file encryption: users can generate encryption keys directly without server interaction. In combination with lazy encryption, efficient and low-latency file uploading to a mobile cloud is realized.

(ii) The random binary-tree-based deduplication algorithm reduces the run time complexity when finding duplicates to $\mathcal{O}(\log n)$, where n is the number of outsourced files in the storage. This makes the scheme much more efficient and scalable, especially considering very large data items being outsourced.

(iii) The use of users' secret keys for file encryption ensures strong data confidentiality even for predictable data, while also guaranteeing data integrity and resistance against collusion attacks.

(iv) Noninteractive file encryption with the support of lazy encryption greatly reduces latency in the file uploading process.

1.1. Contribution. We make several contributions in this paper:

(i) We address the challenge of encrypted deduplication in an MCC environment and propose a novel serverless and efficient encrypted deduplication scheme, called SEED, suitable for this environment

(ii) The security of SEED is rigorously analyzed in terms of data confidentiality, data integrity, and collusion resistance

(iii) The effectiveness of SEED is validated by an extensive analysis of its efficiency and performance

1.2. Organization. The remainder of this paper is organized as follows. In Sections 3 and 4, we present the system model and background knowledge, respectively. In Section 5, we describe the proposed scheme in detail. We analyze the security of the scheme in Section 6 and present a comparative and performance analysis in Section 7. Finally, we conclude the paper in Section 8.

2. Related Work

2.1. Convergent Encryption. CE is a cryptographic algorithm that generates identical ciphertexts from identical plaintexts [6, 7]. In CE, a convergent key k is derived by computing $H(M)$, where $M$is data (or a file) and H is a cryptographic hash function. The ciphertext $C \longleftarrow Enc_k(M)$ is then

computed with conventional symmetric encryption algorithm *Enc* and convergent key k. A given plaintext M will always produce an identical ciphertext C. Bellare et al. [14–16] presented message-locked encryption (MLE), which is a generalized framework for CE, and attempted to formalize security. MLE essentially follows the CE approach in the sense that it derives encryption keys deterministically from M.

Despite the novel nature of encrypted deduplication, CE and MLE are insufficiently secure for two reasons [17]. First, they cannot preserve semantic security due to their deterministic nature. Second, the distribution of message space is the only entropic source of randomness in the convergent key. Thus, the key space is reduced to the message space, which is very small compared to the former. This ultimately renders CE and MLE susceptible to brute-force attacks [8].

Many secure deduplication solutions have been built on CE and MLE. While addressing data confidentiality as a primary goal, these previous solutions also attempted to meet other security goals, such as ownership management [18], authentication [19, 20], authorization [21], reliability [22–24], and access control [9, 25, 26]. Recently proposed lattice-based cryptographic schemes for cloud storage [27, 28] are possible candidates for secure deduplication solutions.

2.2. Server-Aided Encryption. To overcome the weaknesses of CE, it is necessary to strengthen the generation of convergent keys so that the key space has high min-entropy. Several solutions have been proposed to achieve this goal. The approach used in *server-aided encryption* is to generate convergent keys through interacting with key servers. By doing so, the probability distribution of the convergent keys becomes independent of the distribution of message space, and thus brute-force attacks can be mitigated.

DupLESS [8] was the first attempt at server-aided encryption. In this approach, users run an interactive key generation protocol with a key server to compute convergent keys. The protocol operates on RSA-based Oblivious Pseudorandom Function (OPRF) [13], and thus, it guarantees that the convergent keys can be computed without revealing any information about the message or the secret of the key server. In this way, adversaries, such as the MCSP or users, cannot recover plaintext (i.e., messages) with offline brute-force attacks on ciphertext, even if the plaintext is easily predicable.

The security of the DupLESS scheme requires the aid of a key server, which is inherently vulnerable to the single-point-of-failure problem. That is, data confidentiality cannot be retained if the server is compromised.

Subsequent attempts at server-aided encryption have been made to overcome the drawbacks of DupLESS. Miao et al. [11] proposed multiserver-aided encryption, which uses several key servers rather than just one. In this approach, key servers cooperate with each other to process convergent key generation requests. More specifically, convergent keys are generated by executing a threshold blind-signature-based protocol [29] with the aid of the group of key servers. Each key server uses a share of a secret key to generate a partial blind signature for the message a user requested. The partial blind signatures are then combined, and, in turn, a convergent key is computed from the blind signature. Unlike DupLESS, multiserver-aided encryption can resist server-compromise attacks unless the attackers gain access to more than t (i.e., the threshold) key servers.

Another solution, proposed by Duan [10], addressed the single point-of-failure inherent in server-aided encryption. Similar to multiserver-aided encryption, in this approach, multiple entities are involved in key generation using an RSA threshold signature. However, it differs in that the tasks of the key servers are distributed to a number of signers (i.e., a qualified subset of users). A key server participates in the system only during the setup phase: it generates a secret key and disperses shares of the secret key across the signers. Convergent keys can be acquired if more than t signers participate in the interactive key generation protocol. Zhang et al. [30] proposed a server-aided encrypted deduplication scheme for electronic health systems.

The aforementioned schemes achieved the goal of mitigating server-compromise attacks on a DupLESS system. However, all server-aided encryption schemes fundamentally require key servers. The necessity of dedicated servers severely restricts user mobility, limiting its application in MCC environments.

2.3. Serverless Encryption. Another approach has been proposed to achieve high levels of data confidentiality in encrypted deduplication without the need for additional servers.

Liu et al. [31] proposed serverless encryption that uses Password Authenticated Key Exchange (PAKE) [32]. Instead of interacting with key servers, it allows convergent keys to be derived in cooperation with online checkers (i.e., a subset of uploaders) through a PAKE-based protocol. However, despite the advantages of removing the servers, this scheme suffers from lower performance, including high latency, because many PAKE steps are required when conducting file encryption.

Several schemes for serverless encryption use pairing-based cryptography. Abadi et al. [15] proposed a scheme that deviates from MLE by fully randomizing all components of the ciphertexts. A study precedent to SEED [33] is also built on bilinear pairing encryption algorithms to make the ciphertexts indistinguishable from a random distribution.

In these pairing-based schemes, a test algorithm that checks for equality among the ciphertexts is necessary because the ciphertexts are fully randomized. However, deduplication using an equality test algorithm inherently has a linear time complexity with the number of files in the storage. Without a tree-based indexing structure, it seriously degrades the performance of the cloud storage service.

3. System Model and Design Goals

3.1. System Model. In this paper, we consider a general architecture of mobile cloud storage services where multiple mobile users outsource their data to remote storage.

User: this is an entity who owns data (or files (We will use the term "file" and "data" interchangeably in this paper.)) and wishes to outsource the data to the cloud storage. A user who uploaded data is referred to as an uploader: he/she is *the initial uploader* of the file Fif it is the first time that Fhas been uploaded to the storage, or *a subsequent uploader* otherwise.

MCSP: this is an entity equipped with abundant storage and computing resources and provides cloud storage services to mobile users. It has an interest in saving storage costs, so it performs deduplication of the outsourced data.

3.2. Threat Model and Security Goals. We consider honest-but-curious adversaries in our threat model. That is, for assigned tasks, MCSP and users will faithfully perform their work within the system. However, they have an interest in obtaining as much information as possible about the outsourced data, beyond their privileges. Thus, our primary security goal is to prevent them from accessing the plaintext version of encrypted data.

In this study, we consider two types of adversaries: (i) an outside adversary, who makes an effort to learn useful information about the outsourced data by playing the role of a user and (ii) an inside adversary, who may be an honest-but-curious MCSP or intruders that have compromised the storage server. Specifically, we aim to achieve the following security goals in the proposed scheme:

(i) Data confidentiality: no adversary can acquire information from the outsourced data using brute-force attacks unless they obtain the corresponding key

(ii) Data integrity: any valid user should be able to check whether the data downloaded from cloud storage has been kept intact

(iii) Collusion resistance: any adversaries without valid ownership of the data should be blocked from obtaining useful information from the data even if they collude with each other

4. Preliminaries

4.1. Server-Side and Client-Side Deduplication. Data deduplication can be classified into two kinds of approaches according to the location where the deduplication occurs. In server-side deduplication, the MCSP performs deduplication once files have been uploaded to the storage. On the other hand, client-side deduplication is executed on the user's side. That is, before outsourcing a file, a user sends a corresponding tag to the MCSP to check whether the file already exists and, if so, to omit the further upload.

4.2. Bilinear Pairings and Hard Problem

Bilinear Map. Let $\mathbb{G}$ and $\mathbb{G}_T$ be two multiplicative cyclic groups of prime order p. Let g be a generator of $\mathbb{G}$. A bilinear map is an injective function $e: \mathbb{G} \times \mathbb{G} \longrightarrow \mathbb{G}_T$ with the following properties:

(i) *Bilinearity*: for all $u, v \in \mathbb{G}$ and all $a, b \in Z_p^*$, we have $e(u^a, v^b) = e(u, v)^{ab}$

(ii) *Nondegeneracy*: $e(g, g) \neq 1$

(iii) *Computability*: there is an efficient algorithm to compute $e(u, v)$ for $\forall u, v \in \mathbb{G}$

Bilinear Diffie–Hellman (BDH) Problem. Let $a, b, c \in Z_p^*$ be chosen at random and let g be a generator of $\mathbb{G}$. The BDH problem is to compute $e(g, g)^{abc} \in \mathbb{G}_T$ given $g, g^a, g^b, g^c \in \mathbb{G}$ as input. The BDH assumption [34] states that no probabilistic polynomial time algorithm can solve the BDH problem with nonnegligible advantage.

4.3. Random Binary Tree. A binary tree is referred to as a *random binary tree* if it is constructed at random from a probability distribution (e.g., a uniform distribution) of binary trees. A random binary tree of size $n \in N$ is formed in the following way. First, a random permutation $\prod$ of n elements is chosen, and the elements in $\prod$ are added one by one into a binary tree. The addition of elements is similar to the way that elements are inserted into a binary search tree. A root node for a random binary tree is obtained from the first element in $\prod$. Each subsequent element is then evaluated on the tree from the root until it reaches a leaf. The evaluation result $b \in \{\text{Left}, \text{Right}\}$ directs the child node for the next evaluation.

5. Serverless and Efficient Encrypted Deduplication

SEED consists of two building blocks: *file encryption* and *deduplication*. We first present these building blocks in Section 5.1 and then describe a data outsourcing protocol constructed upon them in Section 5.2.

5.1. Building Blocks

5.1.1. File Encryption. We introduce some notations prior to giving details on our file encryption algorithm. Let $\mathbb{G}$ and $\mathbb{G}_T$ be two multiplicative groups with the prime order p, and let $H_1: \{0,1\}^* \longrightarrow \mathbb{G}$ be a hash function family. Let SE_k be a symmetric encryption algorithm with an encryption key $k \in \mathbb{K}$, where $\mathbb{K}$ is a key space of the underlying block cipher (e.g., AES), and let $K: \mathbb{G}_T \longrightarrow \mathbb{K}$ be a key derivation function.

$\langle PK, SK \rangle \longleftarrow$ KeyGen ($\mathcal{J}$). Given global information $\mathcal{J} = \langle p, g \rangle$, this algorithm runs as follows:

(1) Pick a random value $x \longleftarrow Z_p^*$ and compute g^x

(2) Set $PK = g^x$ as its public key and $SK = x$ as its secret key, then return $\langle PK, SK \rangle$

$\langle C, MK, \tau \rangle \longleftarrow$ Encrypt(SK, M). Given a secret key $SK = x$ and a message M, this algorithm runs as follows:

(1) Compute a decryption key $dk = \mu^x$ and a tag $\tau = \nu^x$, where $\mu = H_1(M)$ and $\nu = H_1(\mu)$

(2) Pick a random value $s \longleftarrow \mathbb{G}_T$, and compute $C_1 = SE_k(M)$, where $k = K(s)$

(3) Pick a reencryption key $rk \longleftarrow Z_p^*$, and compute $C_2 = e(\mu^{rk}, \nu) \cdot s$ and $T = (\nu^{x^{-1}})^{rk}$

(4) Return a ciphertext $C = \langle C_1, C_2, T\rangle$, a message-derived key $MK = \langle dk.rk\rangle$, and a tag τ

$T' \longleftarrow \mathrm{ReEnc}(MK, T)$. Given a reencryption key rk in a message-derived key MK and a part of a ciphertext T, this algorithm computes $T' = T^{rk}$ and returns T'.

$\{M, \perp\} \longleftarrow Decrypt(SK, MK, C)$. Given a secret key SK, a message-derived key $MK = \langle dk, rk\rangle$, and a ciphertext $C = \langle C_1, C_2, T\rangle$, this algorithm runs as follows:

(1) If T is not reencrypted (Without a loss of security, we assume that the information about whether E is reencrypted or not is implicitly augmented with the ciphertext C), then recover s by computing.

$$\begin{aligned}\frac{C_2}{e(dk, T)} &= \frac{e(\mu^{rk}, \nu) \cdot s}{e(\mu^x, \nu^{x^{-1}rk})} \\ &= \frac{e(\mu, \nu)^{rk} \cdot s}{e(\mu, \nu)^{rk}} \\ &= s.\end{aligned} \tag{1}$$

(2) If T is reencrypted with another reencryption key rk', recover s by computing.

$$\begin{aligned}\frac{C_2}{e(dk, T^{rk^{-1}})} &= \frac{e(\mu^{rk'}, \nu) \cdot s}{e(\mu^x, \nu^{x^{-1}rk \cdot rk'})^{rk^{-1}}} \\ &= \frac{e(\mu, \nu)^{rk'} \cdot s}{e(\mu, \nu)^{rk'}} \\ &= s.\end{aligned} \tag{2}$$

(3) Compute a symmetric decryption key $\kappa = K(s)$ and recover M by decrypting $C_1 = SE_\kappa(M)$ with κ.

(4) (Integrity check) Compute $dk' = \mu'^x$, where $\mu' = H_1(M)$, and check whether $dk = dk'$. If both values are the same, return M as output. Otherwise, return $\perp$.

$\{\mathrm{True}, \mathrm{False}\} \longleftarrow \mathrm{Test}(\delta_i, \delta_j)$. Parse δ_i and δ_j as $\langle PK_i, \tau_i\rangle$ and $\langle PK_j, \tau_j\rangle$, respectively. Then, given public keys $PK_i = g^{x_i}$ and $PK_j = g^{x_j}$ and tags $\tau_i = \nu^{x_i}$ and $\tau_j = (\nu\imath)^{x_j}$, this algorithm runs as follows:

(1) Check whether the following equation holds.

$$\begin{aligned}&e(PK_i, \tau_j) \stackrel{?}{=} e(PK_j, \tau_i), \\ \Longleftrightarrow\ &e(g^{x_i}, (\nu')^{x_j}) \stackrel{?}{=} e(g^{x_j}, \nu^{x_i}).\end{aligned} \tag{3}$$

(2) If the equation holds, return True. Otherwise, return False.

5.1.2. Deduplication. The performance of deduplication depends on the computational complexity of the algorithm used to find file duplicates in the storage. To achieve efficient deduplication that is as fast as a binary search algorithm with logarithmic complexity, we define a *D-tree*, a binary-tree-based data structure for deduplication. A D-tree is a random binary tree of size n, where n is the number of all the distinct outsourced files in the storage. Each node N_i $(0 \le i \le n-1)$ in a D-tree contains deduplication information δ_i for each outsourced file.

A D-tree is an index structure for the storage of the MCSP. Once file F has been uploaded to the storage, the MCSP checks whether it has a duplicate. For this, it performs a binary search over a D-tree using the Test algorithm given in the previous section. The search path for F is determined at random based on a globally publicized random seed. If a node that contains information δ of F is found, then the MCSP performs deduplication. Otherwise, it creates a new node for F and inserts it at the leaf node on the search path.

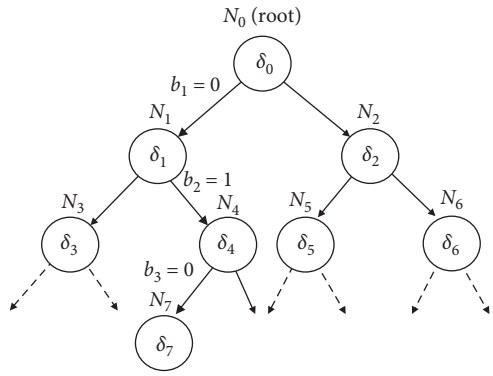
We will introduce here some notation for our deduplication scheme. Let Δ be a D-tree of size n, and let $N_0, \ldots, N_{n-1}$ be its nodes. $\delta_i = \langle PK_j, \tau_j\rangle$ denotes deduplication information assigned to node N_i, where j indicates initial uploader u_j who outsourced τ_j and the corresponding file. Let π be a maximum height of Δ, and let $\psi \in \{0,1\}^\lambda$ be a random seed chosen from a uniform distribution. Let $D = \langle \psi, \pi\rangle$ be global publicly known information. H_2: $\{0,1\}^* \longrightarrow \{0,1\}^\lambda$ denotes a hash function family, and P: $\{0,1\}^* \longrightarrow \{0,1\}$ denotes a digest function. Let $p = (b_1, \ldots, b_\pi)$ be a binary vector of length π, and let $p[i] = b_i \in \{0,1\}$ be the ith element of p. Vector p denotes a search path from a root node on Δ: the bit value of b_i indicates left or right child of the node at the $(i-1)$th level of Δ.

Figure 1 shows an instance of a D-tree of size $n = 7$ and $\pi = 3$ and its storage structure. Nodes in the tree are traversed from a root node N_0 with respect to a search path p, in which the bit information of each element indicates the next child node: bit 0 directs the traversal to the left child and bit 1 to the right child. For example, nodes traversed along a path $p = \{0, 1, 0\}$ include N_0, N_1, N_4, and N_7, with which the corresponding deduplication information $\delta_0, \delta_1, \delta_4$, and δ_7 are sequentially evaluated using the Test algorithm.

Details of the D-tree based deduplication algorithms are given below.

InsertNode(N_i, p). Given a node N_i and a path p, this algorithm inserts N_i at the leaf node of Δ on p.

DeleteNode(N_i). Given a node N_i, this algorithm deletes the node from Δ. If N_i is a non–leaf node, then the deletion is performed by replacing it with one of its child nodes.



(a)

k	Node	Storage	Initial uploader
0	N_0	$\delta_0 \leq PK_1, \tau_1 >, \text{loc}_0$	u_1
1	N_1	$\delta_1 \leq PK_2, \tau_2 >, \text{loc}_1$	u_2
2	N_4	$\delta_4 \leq PK_3, \tau_3 >, \text{loc}_4$	u_3
3	N_7	$\delta_7 \leq PK_4, \tau_4 >, \text{loc}_7$	u_4

loc_k: location to the cipher text C corresponding to δ_k

(b)

Figure 1: D-tree and its storage. (a) D-tree. (b) Storage of nodes on path $P = (b_1, b_2, b_3) = (0, 1, 0)$.

$\{\perp, N_c\} \longleftarrow \text{GetChildNode}(N_i, c)$. Given node N_i and $c \in \{\text{Left}, \text{Right}\}$, this algorithm returns the left or right child node N_c of N_i according to c. If the node does not have N_c, then it returns $\perp$.

$p \longleftarrow D\text{Path}(M, \mathcal{D})$. Given message M and global information $D = \langle \psi, \pi \rangle$, this algorithm runs as described in Algorithm 1. It outputs a path vector $p = (b_1, \ldots, b_\pi)$, where each b_i $(0 \leq i \leq \pi)$ is 0 or 1.

$\langle DuplicatedFound, k \rangle \longleftarrow \text{FindDuplicate}(p, \delta_i)$. Given path vector p of message M and its corresponding deduplication information $\delta_i = \langle PK_i, \tau_i \rangle$, this algorithm runs as follows (the detailed procedure is presented in Algorithm 2).

(1) Get the root node N_0 of a D-tree Δ.

(2) If $N_k = \perp$, return $\langle \text{False}, k \rangle$ (initially $k = 0$). Otherwise, run a test algorithm $b \longleftarrow \text{Test}(\delta_i, \delta_k)$ for the deduplication information δ_k of node N_k.

(3) If b is True, then return $\langle \text{True}, k \rangle$ and halt.

(4) If b is False, then get child node N_c of N_k by running GetChildNode. Choosing a child node depends on $p[k]$: a left child node is selected for $p[k] = 0$, and vice versa. Set $k \longleftarrow k + 1$ and $N_k \longleftarrow N_c$, and then repeat step 2.

Input: M, ψ
Output: $p = (b_1, \cdots, b_\pi)$
(1) $h_0 \longleftarrow H_2(\psi || M)$
(2) $b_0 \longleftarrow 0$
(3) **for** each $i \in [1, \pi]$ **do**
(4) $\quad h_i \longleftarrow H_2(h_i - 1 || b_i - 1)$
(5) $\quad b_i \longleftarrow P(h_i)$
(6) **end for**
(7) **return** $(b_1, \cdots, b_\pi)$

Algorithm 1: D-path.

5.2. Data Outsourcing Protocol

5.2.1. Data Outsourcing in Server-Side Deduplication. We first present the data outsourcing protocol in server-side deduplication. It consists of four operations: system setup, file upload, file download, and file deletion. For clarity, we denote the public key and secret key that belong to user u_i as PK_i and SK_i, respectively. We also denote a message-derived key calculated from SK_i as MK_i. The details of the proposed protocol are given as follows.

System Setup. Given security parameter λ, the system generates public information $\text{Pub} = \langle \mathcal{J}, \mathcal{D} \rangle . \mathcal{J}$. $\mathcal{J}$ consists of the generator g of $\mathbb{G}$ and the order p, and $\mathcal{D}$ consists of the randomly generated integer $\psi \in \{0, 1\}^\lambda$ and the maximum height π of a D-tree Δ. Each user u_i generates a pair of public key PK_i and secret key SK_i by running $\langle PK_i, SK_i \rangle \longleftarrow KeyGen(\mathcal{J})$. Then, PK_i is made public, while SK_i is kept secret.

File Upload. Suppose that a user u_i wishes to upload a file F to the MCSP. u_i performs a file uploading operation as follows:

(1) u_i encrypts F by running $\text{Encrypt}(SK_i, F)$ with his/her secret key SK_i to get its ciphertext $C = \langle C_1, C_2, T_i \rangle$, a message-derived key MK_i, and a tag τ

(2) u_i computes a path vector $p$by running $DPath(F, \psi)$

(3) Then, u_i sends $\langle i, \tau, p, C \rangle$ to the MCSP, where i is the identifier of u_i, and keeps MK_i secret for later use

Once the encrypted file C, as well as its corresponding tag τ and p are uploaded, the MCSP tries to eliminate the duplicate of F by running deduplication as follows:

(1) Given $\langle i, \tau, p \rangle$, the MCSP runs the $\text{FindDuplicate}(p, \delta_i)$ algorithm, where $\delta_i = \langle PK_i, \tau \rangle$.

(2) If the result is $\langle \text{False}, k \rangle$, then C has been previously uploaded to the storage. k is the position of a new node N_k where deduplication information of file F will be assigned. N_k is on p (if $k \geq 1$) or a root node if the D-tree Δ is empty (i.e., $k = 0$). The MCSP inserts

```
Input: p, δ_i
Output: DuplicateFound, k
(1) Get the root node N_0 of a D-tree Δ
(2) DuplicateFound ⟵ False
(3) k ⟵ 0
(4) while N_k ! = ⊥ do
(5)    Get deduplication information δ_k assigned to N_k.
(6)    if Test (δ_i, δ_k) = True then
(7)       DuplicateFound ⟵ True
(8)       break
(9)    else
(10)      if p[k] = = 0 then
(11)         N_c ⟵ GetChildNode(N_k, Left)
(12)      else
(13)         N_c ⟵ GetChildNode(N_k, Right)
(14)      end if
(15)      k ⟵ k + 1
(16)      N_k ⟵ N_c
(17)   end if
(18) end while
(19) return (DuplicateFound, k)
```

Algorithm 2: FindDuplicate.

$\delta_i = \langle PK_i, \tau \rangle$in the position k in Δ and stores C with a link to δ_i. The user u_i is then assigned as the initial uploader of F.

(3) If the result is $\langle \text{True}, k \rangle$, then u_i is a subsequent uploader of F. k indicates a position of node N_k that has stored the deduplication information of F. Hence, the MCSP does not have to store C_1, C_2 in C but T_i. Prior to storing T_i, the MCSP finds the initial uploader u_j assigned to N_k and asks u_j to reencrypt T_i. Upon receipt of the request, u_j computes $T_i' = \text{ReEnc}(MK_j, T_i)$ with his/her own key MK_j and returns T_i'. The MCSP appends T_i' to the end of the stored tuple $C = \langle C_1, C_2, T_j, \ldots, T_l' \rangle$, where l is the identifier of another subsequent uploader.

File Download. User u_i interacts with the MCSP to download an outsourced file F. The details are as follows:

(1) u_i sends a request to download the outsourced file F to the MCSP

(2) Upon receiving the request, the MCSP sends the corresponding ciphertext $C = \langle C_1, C_2, T_i \rangle$ to u_i

(3) Given message-derived key MK_i and secret key SK_i, u_i recovers F by running $\text{Decrypt}(SK_i, MK_i, C)$

(4) If the result is $\perp$, then u_i drops the ciphertext

File Deletion. Upon receiving a deletion request for F from user u_i, the MCSP runs the following steps:

(1) If u_i is the only user who owns the file F, the MCSP removes C in the storage. It also deletes the corresponding node in Δ by running the DeleteNode algorithm.

(2) Otherwise, the MCSP only removes T_i in C.

5.2.2. Data Outsourcing in Client-Side Deduplication. The previously described protocol of SEED is based on server-side deduplication. We can easily modify the protocol to operate in a client-side deduplication mode. Specifically, it can be modified such that instead of fully uploading $\langle i, \tau, C \rangle$, user u_i first sends $\langle i, \tau, T_i \rangle$ to the MCSP. The encrypted files C_1 and C_2 will be uploaded later only if FindDuplicate (PK_i, τ) returns Nil.

Lazy Encryption. SEED takes advantage of *lazy encryption* to further enhance the computational efficiency of the file uploading process in client-side deduplication. Lazy encryption is a novel technique that delays file encryption until the MCSP requests to upload subsequent ciphertexts as a result of the FindDuplicate function. It allows a user to omit the job of file encryption when a duplicate is found in the remote storage. In the file uploading process, the task of encryption (i.e., executing *SE* in the Encrypt algorithm) comprises the majority of computation. Hence, lazy encryption significantly reduces the computational burden of the client. This is a crucial performance factor in mobile devices because it is directly related to a reduction of power consumption.

The lazy encryption technique is enabled in SEED due to the concurrency property of the Encrypt algorithm (in Section 5.1.1). More specifically, in the encryption algorithm, a tag τ can be computed concurrently and independently of the computation of a ciphertext. Figure 2(a) intuitively depicts the concurrent processing of the file encryption. The concurrency property is only found in the proposed scheme. All the existing schemes, including MLE [14] and DupLESS [8], have sequential processing; the encryption and tag generation process are performed inherently in a sequential way (see Figure 2(b)).

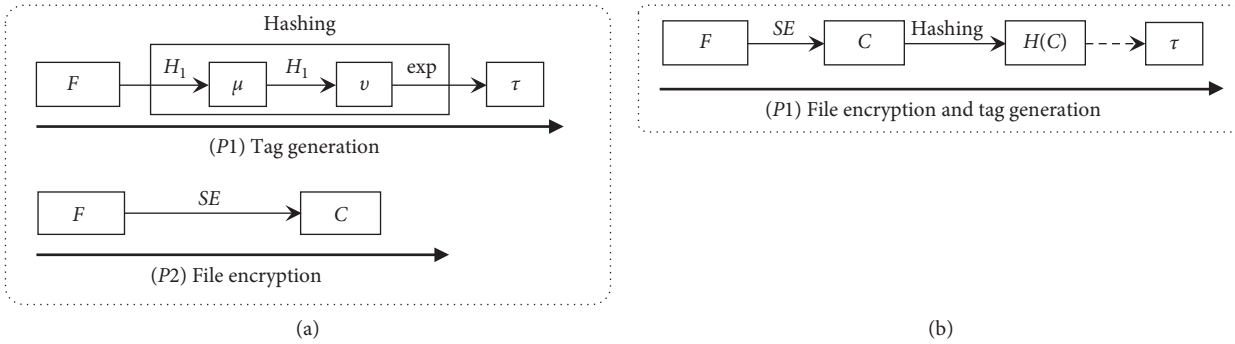


Figure 2: Various types of file encryption processing (*H*_1: hash, *SE*: symmetric encryption, and *exp*: exponentiation). (a) Concurrent processing. (b) Sequential processing.

Side-Channel Prevention. Client-side deduplication is inherently vulnerable to a side-channel attack [35], by which adversaries can infer information about the existence of a specific file in the cloud storage. To defend against such an attack, we use a randomized-threshold approach [35]. In this technique, a randomly chosen threshold t_F ($2 \le t_F \le d$, where d is a security parameter) is assigned to each F in the storage, along with a counter c_F that counts the number of previous uploads of F. Unless c_F reaches t_F, a user will be required to fully upload F as server-side deduplication despite the existence of the file in the storage.

6. Security Analysis

In this section, we analyze the security of SEED regarding data confidentiality, data integrity, and collusion resistance.

6.1. Data Confidentiality. As mentioned in the previous section, our primary security goal with SEED is to guarantee the confidentiality of users' outsourced data. In our threat model, we consider an MCSP that is no longer fully trusted although it is faithful. Therefore, any leakage of users' data should be prevented from adversaries, including the MCSP and unauthorized users. Because our threat model considers various types of attacks from both internal and external adversaries, we analyzed data confidentiality according to these attacks. In the analysis, we assume that all public information, including the public keys of users, are known a priori to the adversaries.

6.1.1. Security against Offline Brute-Force Attacks

Definition 1. An adversary $\mathcal{A}$ runs the following security game: a challenger picks a random bit $b \longleftarrow \{0, 1\}$. $\mathcal{A}$ makes multiple encryption queries with the restriction that only distinct messages are permitted. On each query M, if $b = 1$, the challenger computes the ciphertext C for M and returns it to $\mathcal{A}$. If $b = 0$, the challenger simply returns a random value $r \longleftarrow \{0, 1\}^{|C|}$ to $\mathcal{A}$. At the end of the game, $\mathcal{A}$ outputs $b' \in \{0, 1\}$. An encryption scheme is D-IND$-CPA secure if the advantage $Adv_{\mathcal{A}}^{D-IND\$-CPA} = \Pr[b = b'] - 1/2$ is negligible.

Theorem 1. *SEED is D-IND$-CPA secure in the random oracle model assuming that underlying symmetric encryption algorithm SEis semantically secure and the BDH problem is intractable.*

Proof. In the security game, the adversary $\mathcal{A}$ will be given a correct ciphertext for each query M in the case of $b = 1$. We will show that even in such a case, $\mathcal{A}$ cannot get any information about M from the ciphertext and cannot distinguish from random with nonnegligible advantage. Suppose that the challenger responds to $\mathcal{A}$'s queries as follows: for H_1-random oracle query of M, the challenger picks a random $\mu \in \mathbb{G}$ and returns it to $\mathcal{A}$. For Encrypt oracle query, the challenger returns the corresponding ciphertext $C = \langle C_1, C_2, T\rangle = \langle SE_k(M), e(\mu^{rk}, v) \cdot s, v^{x^{-1}rk}\rangle$ and the tag $\tau = v^x$ to $\mathcal{A}$. □

Because the underlying symmetric encryption algorithm *SE* is semantically secure, the ciphertext C_1 is indistinguishable from random data. That is, because $s \in \mathbb{G}_T$ is chosen at random, the symmetric encryption key κ, which is derived from s, as well as $C_1 = SE_\kappa(M)$, are made pseudorandom. Therefore, $\mathcal{A}$ cannot get any useful information from C_1 except a negligible advantage, unless s is known to $\mathcal{A}$.

Recovering s from C_2 and T is as hard as solving the BDH problem. Suppose that $\mathcal{A}$ can compute s from C_2 and T in polynomial time with nonnegligible probability ε. We can construct an algorithm $\mathcal{B}$ that solves the BDH problem using $\mathcal{A}$: given a BDH instance $\langle g^a, g^b, g^c\rangle$, $\mathcal{B}$ sets up the instance of $\mathcal{A}$ such that $\langle PK, C_2, T\rangle = \langle g^a, R, g^b\rangle$, where R is chosen at random from $\mathbb{G}_T$, and runs $\mathcal{A}$. For an H_1-query of M, $\mathcal{B}$ responds to $\mathcal{A}$ with $\mu = g^c$. From the view of $\mathcal{A}$, the instance is a valid ciphertext of M, such that

$$\begin{aligned} PK &= g^x = g^a, \\ T &= \left(v^{x^{-1}}\right)^{rk} = g^b, \\ C_2 &= e\left(\mu^{rk}, v\right) \cdot s = e(\mu, T)^x \cdot s = e(g, g)^{abc} \cdot s = R, \end{aligned} \tag{4}$$

where s and rk are random values from $\mathbb{G}_T$ and $\mathbb{Z}_p^*$, respectively. If $\mathcal{A}$ terminates and returns $s\prime$ as its output, then $\mathcal{B}$ outputs $O = R/s'$ as the solution of the BDH problem. With nonnegligible probability ε, the output O is the correct

answer of the BDH problem, which contradicts the BDH assumption. Therefore, computing s from C_2 and T is infeasible.

Moreover, because the ciphertexts C_2 and T are blended with two random values s and rk, these ciphertexts are indistinguishable from random data, except with a negligible probability. With regard to a tag $\tau = \nu^x (= H_1(H_1(M))^x)$, $\mathcal{A}$ also cannot distinguish it from random, because for any distinct messages the random oracle H_1 makes ν randomized.

Therefore, SEED makes ciphertexts and tags indistinguishable from random data, which implies that $\mathcal{A}$ has a negligible advantage in winning the security game.

6.1.2. Security against Online Brute-Force Attacks. Now, we analyze the security of SEED against online brute-force attacks. We consider outside adversaries (e.g., unauthorized users) with a dictionary that contains candidates for a file of interest F. The attack proceeds as follows: the adversary repeatedly performs a file upload operation for each candidate $F\prime$ until he/she observes a deduplication event, which indicates the candidate file matches F in the storage.

If the proposed scheme is run under the mode of server-side deduplication, such an attack cannot succeed, this is because all candidates in the dictionary will eventually be sent to the MCSP during the operation, and thus the adversary can infer no information about whether deduplication takes place. In the case of client-side deduplication, the uploading of a certain file may be omitted if it already exists in the storage, which may give information to the adversary. However, the randomized-threshold strategy makes the adversary fully upload the file even if it exists in the storage and thus obfuscates the information about the file. As analyzed in [35], the adversary cannot obtain the information with probability $1 - 1/(d-1)$, where d is a security parameter.

6.2. Data Integrity. The integrity of outsourced data can be compromised by data corruption due to defects in the storage system or adversaries' intentional attacks. SEED provides users with the ability to detect alteration in the outsourced data easily. Say that a user has downloaded an outsourced ciphertext $C = \langle C_1, C_2, T\rangle$ from the MCSP. While running the Decrypt algorithm, the user can restore the plain data F' from the ciphertext and then compute $dk' = \mu\prime^{SK} = H_1(F')^{SK}$. If dk' and a decryption key $dk (= \mu^{SK})$ are different, Decrypt outputs $\perp$, and the user knows that the outsourced file has been modified. Notice that the probability of Decrypt yielding an output other than $\perp$ is negligible for $F \neq F'$, thanks to the collision-resistant property of the cryptographic hash function H_1. Thus, SEED offers an integrity model that allows users to validate the outsourced data effectively.

6.3. Collusion Resistance. SEED also provides security against any collusion attacks. Let us consider the colluding of unauthorized users who do not have valid ownership of file of interest F. Although they have access to ciphertext C of the file, they need the correct decryption key $dk (= \mu^x)$ to decrypt the ciphertext. Suppose that the colluding users have obtained sufficiently many decryption keys for other files. Even with these decryption keys, it is impossible to compute the correct decryption key dk for F unless they know both μ and secret key x.

We also consider an attack in which unauthorized users collude with an MCSP. In addition to decryption keys for other files, they would have access to ciphertexts other than C on the storage. However, because other ciphertexts contain no information about F, these adversaries learn nothing about F. This is the same as in the former case that requires the adversary to compute the correct decryption key dk of F to succeed in the attack. Therefore, the proposed scheme resists attacks by colluding adversaries.

7. Evaluation

7.1. Comparative Analysis. We comparatively analyzed secure deduplication schemes regarding attack resistance, mobility support, file encryption, and deduplication cost. The result is summarized in Table 1.

CE (or MLE) has the cheapest computational cost among deduplication schemes, because any math operations, such as exponentiation or group multiplication, are not required to perform file encryption. However, because of its weak security against brute-force attacks, this scheme cannot guarantee strong confidentiality to the outsourced data. This implies that CE is also vulnerable to server-compromise attacks, because attackers who compromised cloud servers can easily revert CE ciphertexts to plaintexts by brute-force recovery.

Server-aided encryption schemes achieve resistance against brute-force attacks using an OPRF protocol (and its variants) with key servers. However, they also have vulnerability to server-compromise attacks. This is because if one of the key servers is compromised and a secret key is leaked from the server, then the security of the whole system is downgraded to the level of CE. This implies that it fails to guarantee strong confidentiality of outsourced data. Several works by Miao et al. [11] and Duan [10] tried to alleviate the risk of such attacks. However, these approaches still fail if more than k (i.e., threshold) servers are compromised.

Besides, in server-aided encryption, the cost of key computation is larger than in other solutions, and clients are requested to interact with key servers for the generation of convergent keys. This inevitably adds a nonnegligible latency to file outsourcing operations. Such intrinsic latency and the need for key servers, which usually reside in central data centers, make server-aided encryption solutions less attractive in MCC environments, where the support of low-latency service and mobility is critically important.

Liu et al.'s scheme [31] eliminates the need for key servers. Instead of OPRF, this scheme uses a PAKE protocol to achieve security against brute-force attacks. The lack of additional servers inherently leads to improved security that prevents server-compromise attacks. In this scheme, however, executing many PAKE protocols with online checkers

Table 1: Comparison of secure deduplication schemes.

		CE (MLE)	Server-aided encryption			Serverless encryption		
			DupLESS	Miao et al.	Duan et al.	Liu et al.	Abadi et al.	SEED
Attack resistance	Brute-force attack	×	○	○	○	○	○	○
	Server-comp. attack	×	×	Δ	Δ	○	○	○
Key server requirement		×	○	○	○	×	×	×
File encryption cost	Key computation	H	$H+2M+2E$	$H+2\kappa+2E$	$H+(2+\kappa)$ $M+2E$	$H+2HE$	H	$H+E$
	Key computation ($\leftrightarrow$)	—	OPRF	κOPRF	κOPRF	cPAKE	—	—
	Ciphertext computation	SE	SE	$H+SE$	$H+SE$	SE	$S(H+SE+M+E+P)$	$SE+3E+P$
	Tag computation	H	H	H	H	H	$2E$	E
Lazy encryption		NO	NO	NO	NO	NO	NO	Yes
Deduplication cost	Time complexity	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(n)$	$O(\log n)$

H: file hash computation, SE: symmetric encryption, M: group multiplication, E: group exponentiation, P: pairing, HE: homomorphic encryption, k: number of key servers, c: number of online checkers,$\leftrightarrow$: interactive protocol between user and other entities, s: number of shares, and n: number of files in cloud storage.

(i.e., users) is mandatory for each file encryption. Like server-aided encryption, this will incur high latency in performing file encryption, which degrades the effectiveness of the scheme and makes it unsuitable for MCC environments.

Abadi et al.'s scheme [15] requires neither additional servers nor interactive protocols with any entities in file encryption. However, full randomization in file encryption incurs an extremely high computational cost for ciphertext computation. In addition, randomized ciphertexts consequently lose ordering information that is necessary to allow deduplication using a tree-based index structure. Jiang et al. [36] addressed the problem of Abadi et al.'s scheme and proposed a method that achieves logarithmic complexity in searching duplicate files for the fully randomized deduplication. Their method uses a tree-based data structure called a *decision tree*, which is similar to a D-tree. Despite sharing the underlying tree-based approach, there are significant differences; in Jiang et al.'s scheme, a user is required to interactively query the cloud server for each node on a path to find duplicates, while this can be achieved in a noninteractive manner in the proposed scheme.

As analyzed in Section 6.1, SEED guarantees strong confidentiality against brute-force attacks without using any additional key servers. Even if the MCSP is compromised, plain data cannot be recovered because the success probability of a brute-force attack is negligible. Therefore, SEED offers further security against server-compromise attacks. Although more math operations for ciphertext computation are needed than in server-aided encryption schemes, any interactions with servers are unnecessary while conducting file encryption. In addition, SEED achieves low latency in file encryption because it supports the novel property of lazy encryption, which is infeasible for other client-side deduplication schemes that require full ciphertext computation for tag generation. Using a random binary tree reduces the complexity of the deduplication algorithm to $\mathcal{O}(\log n)$, which makes SEED much more efficient and scalable in MCC environments.

7.2. Experiments. To evaluate the computational efficiency, we implemented SEED and other deduplication schemes using Charm [37], a Python-based framework for prototyping cryptosystems. Charm provides useful math operations, such as group multiplication, exponentiation, and bilinear pairing, through Python wrap-up modules of the native C libraries GNU Multiple Precision Arithmetic Library (GMP) and Paring-Based Crypto Library (PBC). Therefore, the performance overhead caused by the use of Python is limited to less than 1% [37]. We selected the SS501 curve in our experiment, which is a supersingular elliptic curve with symmetric Type 1 pairing. We chose SHA-256 as the cryptographic hash function and AES-CBC with 128-bit keys as the symmetric block cipher algorithm.

Our implementation consists of two modules: a client-side program simulating a file-uploading user and a server-side program simulating MCSP, which oversees deduplication. In all our experiments, the client-side program was executed on a PC with an Intel Core i7-4770 3.4 GHz CPU and 4 GB of RAM, and the server-side program was executed on a server with an Intel Xeon E5-2676 2.4 GHz CPU and 8 GB of RAM. Ubuntu 14.04 LTS (64 bits) was installed and run on both the PC and the server. For server-aided encryption schemes, we used a LAN with a 100 Mbps Ethernet link to execute interactive protocols with a remote key server.

7.3. File Encryption. In secure deduplication schemes, file encryption makes up the majority of a user's computational burden for the file uploading phase. Therefore, we measured the execution time of file encryption in SEED and other schemes. For server-aided encryption, we chose DupLESS as

a comparative scheme because its computational cost is the cheapest of its kind [17].

We conducted the experiment for both deduplication architectures (i.e., client-side and server-side deduplication) with sample files whose size varied from 1 MB to 1 GB. Regarding client-side deduplication, we assumed that deduplication always happens for all the sample files. For each experiment, the measurement was repeated 1,000 times. The results of the experiments are shown in Figure 3. The term "Execution time" on the y-axis refers to the elapsed time to compute a ciphertext C from a corresponding file F. For client-side deduplication, it actually means the required time to generate a tag τ, because of the above assumption.

As shown in Figure 3, SEED shows better computational performance than DupLESS [8], Liu et al.'s scheme [31], and Abadi et al.'s scheme [15], which essentially require large computational tasks or high-latency interactions with remote entities during file encryption. Among server-side deduplication schemes, CE shows the least execution time because of its simplicity. However, in the case of being operated as client-side deduplication (Figure 3(b)), SEED shows the best computational performance owing to the novel property of lazy encryption. This is because the encryption of a file (i.e., SE operation) can be omitted when deduplication takes place. All other schemes, including CE, must generate a full ciphertext whatever the deduplication result is, because the ciphertext is required for computing the corresponding tag. Hence, those schemes in client-side deduplication showed no difference in the performance of file encryption with server-side deduplication.

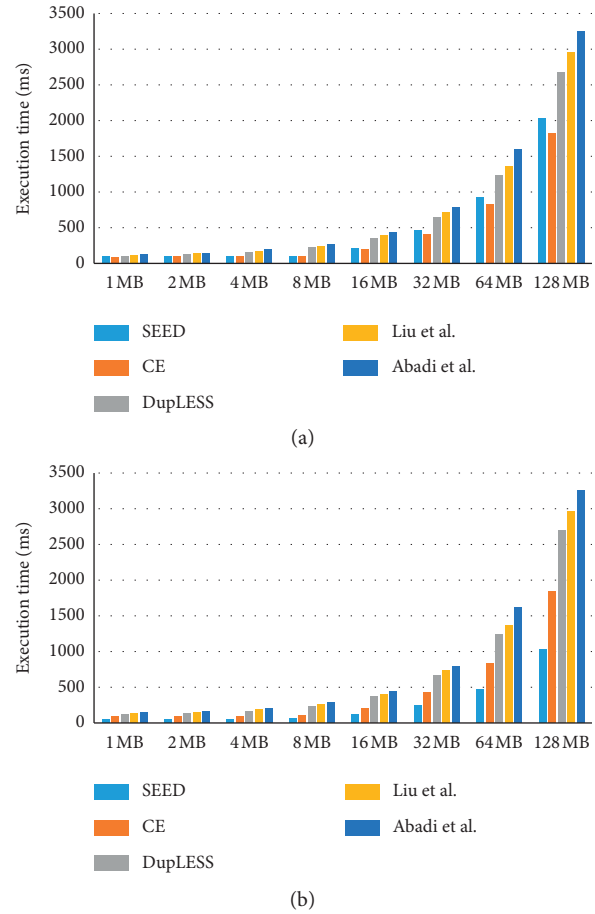


Figure 3: Execution time for file encryption. (a) Server-side deduplication. (b) Client-side deduplication.

7.4. Deduplication. In our second experiment, we measured the computational efficiency of the D-tree-based deduplication algorithm. The data set for the experiment consisted of files sampled from Windows system files, media files, Office files, and so on. The number of files varied from 100 to 20,000. The maximum height of the D-tree was set to be $\pi = 15$.

For the comparison, we also implemented the deduplication algorithms of other schemes. We chose a red-black tree as the indexing structure of our implementations for CE, DupLESS, and Liu et al.'s scheme. A red-black tree is a type of self-balancing binary search tree that guarantees searching in $\mathcal{O}(\log n)$ time in the average case [38]. For Abadi et al.'s scheme, we used sequential search, because the equality test algorithm does not support a binary search tree.

Figure 4 shows the result of the experiment. The term "Number of test operations" refers to the number of operations to test equality between ciphertexts for each deduplication. For SEED, it means the number of executions of the Test algorithm. Because D-tree allows binary search for tags, the number of Test executions for each data set is almost the same as in the other schemes using red-black trees. SEED achieves 2-3 orders of magnitude higher performance (i.e., fewer equality-test operations) than Abadi et al.'s scheme.

We also measured the actual elapsed time during the execution of deduplication algorithms. Figure 5 presents the execution time to complete deduplication for each data set. SEED needed slightly more execution time than the other schemes that use red-black trees, because the Test algorithm includes bilinear pairing operations, which incur high computational costs. Despite the computational overhead, however, the execution time does not exceed 150 ms even for the data set with the maximum number of files. We believe that the computational overhead can be further reduced using high-performance computing technologies, such as distributed and concurrent processing.

8. Discussion

8.1. Reliability of the Initial Uploader. In the proposed scheme, an initial uploader contributes to subsequent file upload processes by reencrypting a part of a ciphertext. Because the reencryption is crucial for subsequent uploaders to access the encrypted content, it is required that the initial uploader remains online to serve requests without interruption. In a mobile environment, for which the proposed scheme is intended, mobile devices are likely to be connected to the Internet most of the time. Hence, we reasonably assume that the reliability of the participation of an initial uploader (i.e., a mobile device) will be acceptable in most cases.

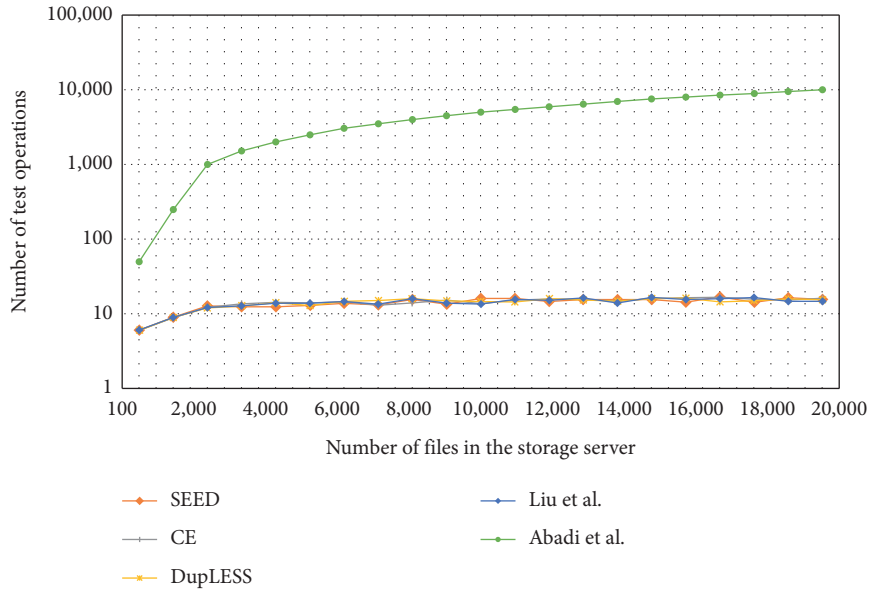


Figure 4: Number of test operations for deduplication ($\pi = 15$).

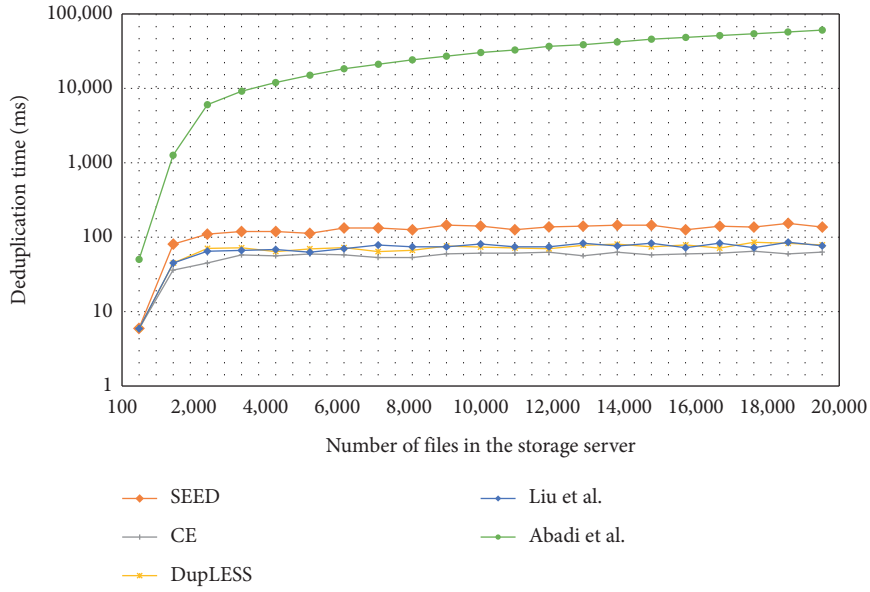


Figure 5: Execution time for deduplication ($\pi = 15$).

However, we should consider the possibility that the initial uploader might not be available due to various reasons (e.g., temporary loss of the connection). For the sake of more reliable service, we may relax the protocol, so that the first N ($N \geq 1$) users that uploaded a file are regarded as the initial uploaders. Subsequent uploaders will be able to successfully conduct the file upload process if at least one initial uploader responds to the reencrypting request.

We analyzed the reliability of file uploading for subsequent uploaders with regard to the number of initial uploaders. Consider the case where an initial uploader is not available when a reencryption request has been sent. Suppose that this event happens with a probability p independently from each other. Then, the probability that at least one initial uploader will successfully respond to the request is $1 - p^N$. Figure 6 shows the probability with regard to N and p. Commercial cloud services such as AWS and Azure usually provide an Service-Level Agreement (SLA) that guarantees more than 95% in terms of service availability. With this information, we can choose the appropriate parameter N. For instance, we choose $N = 1$ for the case of $p = 0.05$ and $N = 2$ for $p = 0.1$.

8.2. Storage Overhead due to Ciphertext Expansion. The proposed scheme relies on a bilinear pairing-based

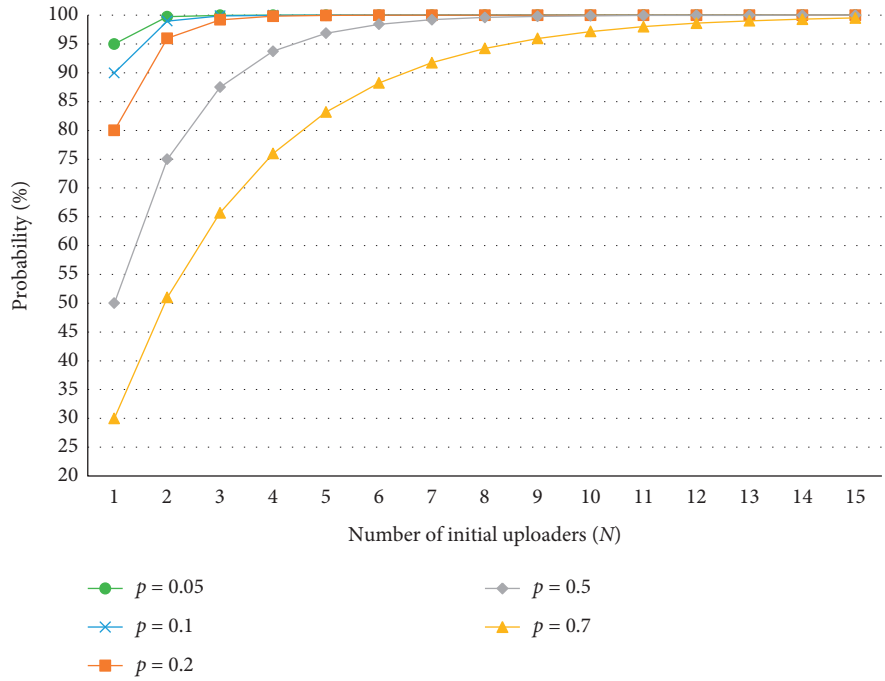


Figure 6: Reliability of file uploading for subsequent uploaders with regard to the number of initial uploaders (N).

cryptosystem. Therefore, a ciphertext generated under the proposed scheme consists of several components whose size is directly related to the pairing. More specifically, the Encrypt algorithm, described in Section 5.1.1, generates a ciphertext $C = \langle C_1, C_2, T\rangle$, among which C_2 is an element of $\mathbb{G}_T$ and T is an element of $\mathbb{G}$, where $\mathbb{G}$ and $\mathbb{G}_T$ are multiplicative groups that form a pairing e: $\mathbb{G} \times \mathbb{G} \longrightarrow \mathbb{G}_T$. Hence, the ciphertext size expands exactly by $|\mathbb{G}| + |\mathbb{G}_T|$. Regarding the storage overhead for the cloud service provider (i.e., MCSP), the ciphertext expansion may cause a certain level of performance degradation.

However, the storage overhead can be minimized due to the deduplication feature of the proposed scheme. That is, it is not necessary to store all the ciphertext components for deduplicated files in the storage. As described in Section 5.2.1, only T in $C = \langle C_1, C_2, T\rangle$ needs to be stored in the case where a duplicate file is found.

9. Conclusion

In this paper, we addressed the problem of deduplication over encrypted data in MCC environments by proposing SEED, a serverless and efficient encrypted deduplication scheme. The novelty of SEED originates from the elimination of key servers, which severely restrict user mobility, while not losing effective data confidentiality. The computational efficiency of file encryption is achieved through noninteractive file encryption and support for lazy encryption. As a result, SEED offers efficient, low-latency file uploading for mobile cloud storage.

Furthermore, a D-tree-based deduplication algorithm successfully reduces the time complexity of deduplication to $\mathcal{O}(\log n)$. This makes SEED much more efficient and scalable, even in the case of large data items being outsourced in the storage.

The security of SEED was rigorously analyzed in this paper, and it was shown that the proposed scheme strongly guarantees security against brute-force attacks without the help of any key servers. The analysis showed that other desired security properties, such as data integrity and collusion resistance, were also achieved by SEED.

Extensive comparative analysis and experiments were conducted to evaluate the performance of SEED. We showed that SEED has advantages in security and efficiency compared to other encrypted deduplication solutions.

Data Availability

The experimental results used to support the findings of this study are available from the corresponding author upon request.

Conflicts of Interest

The authors declare that they do not have any conflicts of interest regarding the publication of this paper.

Acknowledgments

This work was extended from the poster presented at IEEE CloudCom [33]. This research was conducted under a Research Grant from Kwangwoon University in 2020. This work was supported by the Institute of Information & Communications Technology Planning & Evaluation (IITP) grants funded by the Korean Government (MSIT) (No. 2019-0-00533, Research on CPU Vulnerability Detection and Validation), (No. 2019-0-00426, Development of Active

Kill-Switch and Biomarker-Based Defense System for Life-Threatening Internet of Things Medical Devices), and (No. 2020-0-00325, Traceability Assurance Technology Development for Full Lifecycle Data Safety of Cloud Edge).

References

[1] A. U. R. Khan, M. Othman, S. A. Madani, and S. U. Khan, "A survey of mobile cloud computing application models," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 1, pp. 393–413, 2014.

[2] F. Liu, P. Shu, H. Jin et al., "Gearing resource-poor mobile devices with powerful clouds: architecture, challenges and applications," *IEEE Wireless Communications Magazine, Special Issue on Mobile Cloud Computing*, vol. 20, no. 3, pp. 14–22, 2013.

[3] E. Ahmed and M. H. Rehmani, "Mobile edge computing: opportunities, solutions, and challenges," *Future Generation Computer Systems*, vol. 70, 2017.

[4] S. Abolfazli, Z. Sanaei, E. Ahmed, A. Gani, and R. Buyya, "Cloud-based augmentation for mobile devices: motivation, taxonomies, and open challenges," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 1, pp. 337–368, 2014.

[5] N. Mandagere, P. Zhou, M. A. Smith, and S. Uttamchandani, "Demystifying data deduplication," in *Proceedings of the Acm/ifip/usenix Middleware '08 Conference Companion (COMPANION'08)*, pp. 12–17, Leuven, Belgium, December 2008.

[6] J. R. Douceur, A. Adya, W. J. Bolosky, D. Simon, M. Theimer, and P. Simon, "Reclaiming space from duplicate files in a serverless distributed file system," in *Proceedings of the 22nd International Conference on Distributed Computing Systems (ICDCS 2002)*, pp. 617–624, Vienna, Austria, July 2002.

[7] M. W. Storer, K. Greenan, D. D. E. Long, and E. L. Miller, "Secure data deduplication," in *Proceedings of the 4th ACM International Workshop on Storage Security and Survivability (STORAGESS'08)*, pp. 1–10, Fairfax, VA, USA, October 2008.

[8] M. Bellare, S. Keelveedhi, and T. Ristenpart, "DupLESS: server-aided encryption for deduplicated storage," in *Proceedings of USENIX Security Symposium*, pp. 179–194, Washington, DC, USA, August 2013.

[9] H. Tang, "Enabling ciphertext deduplication for secure cloud storage and access control," in *Proceedings of the 2016 ACM Asia Conference on Computer and Communications Security (ASIACCS'16)*, pp. 59–70, Xi'an, China, May 2016.

[10] Y. Duan, "Distributed key generation for encrypted deduplication," in *Proceedings of the 6th ACM Workshop on Cloud Computing Security (CCSW'14)*, pp. 57–68, Scottsdale, AZ, USA, November 2014.

[11] M. Miao, J. Wang, H. Li, and X. Chen, "Secure multi-server-aided data deduplication in cloud computing," *Pervasive and Mobile Computing*, vol. 24, pp. 129–137, 2015.

[12] Y. Shin, D. Koo, J. Yun, and J. Hur, "Decentralized server-aided encryption for secure deduplication in cloud storage," *IEEE Transactions on Services Computing*, pp. 1–13, 2019.

[13] M. Naor and O. Reingold, "Number-theoretic constructions of efficient pseudo-random functions," *Journal of the ACM*, vol. 51, no. 2, pp. 231–262, 2004.

[14] M. Bellare, S. Keelveedhi, and T. Ristenpart, "Message-Locked Encryption and secure deduplication," in *Advances in Cryptology-Eurocrypt*, T. Johansson and P. Q. Nguyen, Eds., Vol. 7881, Springer, Berlin, Heidelberg, 2013.

[15] 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.

[16] M. Bellare and S. Keelveedhi, "Interactive message-locked encryption and secure deduplication," *Public-Key Cryptography-PKC*, vol. 9020, pp. 1–29, 2015.

[17] Y. Shin, D. Koo, and J. Hur, "A survey of secure data deduplication schemes for cloud storage systems," *ACM Computing Surveys*, vol. 49, no. 4, pp. 1–38, 2017.

[18] J. Hur, D. Koo, Y. Shin, and K. Kang, "Secure data deduplication with dynamic ownership management in cloud storage," *IEEE Transactions on Knowledge and Data Engineering*, vol. 28, no. 1, pp. 1–14, 2016.

[19] L. González-Manzano and A. Orfila, "An efficient confidentiality-preserving Proof of Ownership for deduplication," *Journal of Network and Computer Applications*, vol. 50, pp. 49–59, 2015.

[20] J. Xu, E.-C. Chang, and J. Zhou, "Leakage-Resilient client-side deduplication of encrypted data in cloud storage," in *Proceedings of the 8th ACM SIGSAC Symposium on Information, Computer and Communications Security (ASIACCS'13)*, p. 195, Hangzhou, China, May 2013.

[21] J. Li, Y. K. Li, X. Chen, P. P. C. Lee, and W. Lou, "A hybrid cloud approach for secure authorized deduplication," *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, no. 5, pp. 1206–1216, 2015.

[22] M. Li, C. Qin, and P. P. C. Lee, "CDStore: toward reliable, secure, and cost-efficient cloud storage via convergent dispersal," in *Proceedings of the 2015 USENIX Annual Technical Conference (ATC'15)*, pp. 1–20, Santa Clara, CA, USA, July 2015.

[23] M. Li, C. Qin, P. P. C. Lee, and J. Li, "Convergent dispersal: toward storage-efficient security in a cloud-of-clouds," in *Proceedings of the 6th USENIX Conference on Hot Topics in Storage and File Systems (HOTSTORAGE'14)*, Philadelphia, PA, USA, June 2014.

[24] J. Li, X. Chen, M. Li, J. Li, and P. P. C. Lee, "Secure deduplication with efficient and reliable convergent key management," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 6, pp. 1615–1625, 2014.

[25] X. Lou, R. Lu, J. Shao, X. Tang, and A. Ghorbani, "Achieving efficient secure deduplication with user-defined access control in cloud," *IEEE Transactions on Dependable and Secure Computing*, pp. 1–5, 2020.

[26] Q. Huang, Z. Zhang, and Y. Yang, "Privacy-preserving media sharing with scalable access control and secure deduplication in mobile cloud computing," *IEEE Transactions on Mobile Computing*, p. 1, 2020.

[27] X. Zhang, C. Xu, H. Wang, Y. Zhang, and S. Wang, "Lattice-based forward secure public-key encryption with keyword search for cloud-assisted industrial Internet of things," *IEEE Transactions on Dependable and Secure Computing*, 2019.

[28] X. Zhang, H. Wang, and C. Xu, "Identity-based key-exposure resilient cloud storage public auditing scheme from lattices," *Information Sciences*, vol. 472, pp. 223–234, 2019.

[29] W. Cui, Y. Xin, Y. Yang, and X. Niu, "A new blind signature and threshold blind signature scheme from pairings," in *Proceedings of International Conference on Computational Intelligence and Security Workshops (CISW 2007)*, pp. 699–702, Harbin, China, December 2007.

[30] 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.

[31] J. Liu, N. Asokan, and B. Pinkas, "Secure deduplication of encrypted data without additional independent servers," in *Proceedings of the 22th ACM SIGSAC Conference on Computer and Communications Security (CCS 2015)*, pp. 874–885, Denver, CO, USA, October 2015.

[32] S. M. Bellovin and M. Merritt, "Encrypted key exchange: password-based protocols secure against dictionary attacks," in *Proceedings of the IEEE Symposium on Security and Privacy (S&P 1992)*, pp. 72–84, Oakland, CA, USA, May 1992.

[33] Y. Shin, D. Koo, J. Yun, and J. Hur, "SEED: enabling serverless and efficient encrypted deduplication for cloud storage," in *Proceedings of the 8th IEEE International Conference on Cloud Computing Technology and Science (CLOUDCOM 2016)*, pp. 482–487, Luxembourg City, Luxembourg, December 2016.

[34] D. Boneh and M. Franklin, "Identity-based encryption from the weil pairing," *SIAM Journal on Computing*, vol. 32, no. 3, pp. 586–615, 2003.

[35] D. Harnik, B. Pinkas, and A. Shulman-Peleg, "Side channels in cloud services: deduplication in cloud storage," *IEEE Security & Privacy Magazine*, vol. 8, no. 6, pp. 40–47, 2010.

[36] T. Jiang, X. Chen, Q. Wu, J. Ma, W. Susilo, and W. Lou, "Secure and efficient cloud data deduplication with randomized tag," *IEEE Transactions on Information Forensics and Security*, vol. 12, no. 3, pp. 532–543, 2017.

[37] J. A. Akinyele, C. Garman, I. Miers et al., "Charm: a framework for rapidly prototyping cryptosystems," *Journal of Cryptographic Engineering*, vol. 3, no. 2, pp. 111–128, 2013.

[38] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, "Red-black trees," in *Introduction to Algorithms*, pp. 273–301, MIT Press, Cambridge, MA, USA, second edition, 2001.