Secure Data Deduplication with Dynamic Ownership Management in Cloud Storage

Junbeom Hur, Dongyoung Koo, Youngjoo Shin, and Kyungtae Kang

Abstract—In cloud storage services, deduplication technology is commonly used to reduce the space and bandwidth requirements of services by eliminating redundant data and storing only a single copy of them. Deduplication is most effective when multiple users outsource the same data to the cloud storage, but it raises issues relating to security and ownership. Proof-of-ownership schemes allow any owner of the same data to prove to the cloud storage server that he owns the data in a robust way. However, many users are likely to encrypt their data before outsourcing them to the cloud storage to preserve privacy, but this hampers deduplication because of the randomization property of encryption. Recently, several deduplication schemes have been proposed to solve this problem by allowing each owner to share the same encryption key for the same data. However, most of the schemes suffer from security flaws, since they do not consider the dynamic changes in the ownership of outsourced data that occur frequently in a practical cloud storage service. In this paper, we propose a novel server-side deduplication scheme for encrypted data. It allows the cloud server to control access to outsourced data even when the ownership changes dynamically by exploiting randomized convergent encryption and secure ownership group key distribution. This prevents data leakage not only to revoked users even though they previously owned that data, but also to an honest-but-curious cloud storage server. In addition, the proposed scheme guarantees data integrity against any tag inconsistency attack. Thus, security is enhanced in the proposed scheme. The efficiency analysis results demonstrate that the proposed scheme is almost as efficient as the previous schemes, while the additional computational overhead is negligible.

Index Terms—Deduplication, cloud storage, encryption, proof-of-ownership, revocation.

1 Introduction

Cloud computing provides scalable, low-cost, and location-independent online services ranging from simple backup services to cloud storage infrastructures. The fast growth of data volumes stored in the cloud storage has led to an increased demand for techniques for saving disk space and network bandwidth. To reduce resource consumption, many cloud storage services, such as Dropbox [1], Wuala [2], Mozy [3], and Google Drive [4], employ a deduplication technique, where the cloud server stores only a single copy of redundant data and provides links to the copy instead of storing other actual copies of that data, regardless of how many clients ask to store the data. The savings are significant [5], and reportedly, business applications can achieve disk and bandwidth savings of more than 90% [6]. However, from a security perspective, the shared usage of users' data raises a new challenge.

As customers are concerned about their private data, they may encrypt their data before outsourcing in order to protect data privacy from unauthorized outside adversaries, as well as from the cloud service provider [7],[8],[9]. This is justified by current security trends and numerous industry regulations such as PCI DSS [10]. However, conventional encryption makes deduplication impossible for the following reason. Deduplication techniques take advantage of data similarity to identify the same data and reduce the storage space. In contrast, encryption algorithms randomize the encrypted files in order to make ciphertext indistinguishable from theoretically random data. Encryptions of the same data by different users with different encryption keys results in different ciphertexts, which makes it difficult for the cloud server to determine whether the plain data are the same and deduplicate them. Say a user Alice encrypts a file M under her secret key sk_A and stores its corresponding ciphertext C_A. Bob would store C_B, which is the encryption of M under his secret key sk_B. Then, two issues arise: (1) how can the cloud server detect that the underlying file M is the same, and (2) even if it can detect this, how can it allow both parties to recover the stored data, based on their separate secret keys?

- *J. Hur is with the Department of Computer Science and Engineering, Korea University, Seoul, 136-701, Republic of Korea. E-mail: jbhur@korea.ac.kr*
- *D. Koo is with the Department of Computer Science and Engineering, Korea University, Seoul, 136-701, Republic of Korea. E-mail: dykoo@nslab.kaist.ac.kr*
- *Y.Shin is with the Department of Convergence Security, Kyonggi University, Suwon, 16227, Republic of Korea. E-mail: s.youngjoo@gmail.com*
- *K. Kang is with the Department of Computer Science and Engineering, Hanyang University, Ansan, 426-791, Republic of Korea. E-mail: ktkang@hanyang.ac.kr*

Straightforward client side encryption that is secure against a chosen-plaintext attack with randomly chosen encryption keys prevents deduplication [11],[12]. One naive solution is to allow each client to encrypt the data with the public key of the cloud storage server. Then, the server is able to deduplicate the identified data by decrypting it with its private key pair. However, this solution allows the cloud storage server to obtain the outsourced plain data, which may violate the privacy of the data if the cloud server cannot be fully trusted [13],[14].

Convergent encryption [15] resolves this problem effectively. A convergent encryption algorithm encrypts an input file with the hash value of the input file as an encryption key. The ciphertext is given to the server and the user retains the encryption key. Since convergent encryption is deterministic[1], identical files are always encrypted into identical ciphertext, regardless of who encrypts them. Thus, the cloud storage server can perform deduplication over the ciphertext, and all owners of the file can download the ciphertext (after the proof-of-ownership (PoW) process optionally) and decrypt it later since they have the same encryption key for the file. Convergent encryption has long been studied in commercial systems and has different encryption variants for secure deduplication [8],[16],[17],[18], which was formalized as message-locked encryption later in [20]. However, convergent encryption suffers from security flaws with regard to tag consistency and ownership revocation.

As an example of the tag consistency attack issue, suppose Alice and Bob have the same data M, and Alice generates ciphertext C_A from M, and then maliciously generates another ciphertext C'_A from $M'(\neq M)$. Next, she uploads C'_A with an honestly generated tag $\mathcal{T}(C_A) = H(M)$ for a cryptographic hash function H, which plays the role of data index. When Bob generates ciphertext C_B from M and tries to upload C_B, the cloud server checks $\mathcal{T}(C_A) = \mathcal{T}(C_B)$. Then, it deletes C_B and keeps only C'_A. Afterwards, when Bob downloads and decrypts it, the data would be M', not M, which means the integrity of his data has been compromised. Recently, message-locked encryption (MLE) [20] and leakage-resilient deduplication [19] schemes have been proposed to solve this problem by introducing additional integrity check phase for decrypted data.

In the case of ownership revocation, suppose multiple users have ownership of a ciphertext outsourced in cloud storage. As time elapses, some of these users may request the cloud server to delete or modify their data, and then, the server deletes the ownership information of the users from the ownership list for the corresponding data. Then, the revoked users should be prevented from accessing the data stored in the cloud storage after the deletion or modification request (forward secrecy). On the other hand, when a user uploads data that already exist in the cloud storage, the user should be deterred from accessing the data that were stored before he obtained the ownership by uploading it (backward secrecy)[2]. These dynamic ownership changes may occur very frequently in a practical cloud system, and thus, it should be properly managed in order to avoid the security degradation of the cloud service. However, the previous deduplication schemes could not achieve secure access control under a dynamic ownership changing environment, in spite of its importance to secure deduplication, because the encryption key is derived deterministically and rarely updated after the initial key derivation. Therefore, for as long as revoked users keep the encryption key, they can access the corresponding data in the cloud storage at any time, regardless of the validity of their ownership. This is the problem we attempt to solve in this study.

1.1 Contribution

We propose a deduplication scheme over encrypted data. The proposed scheme ensures that only authorized access to the shared data is possible, which is considered to be the most important challenge for efficient and secure cloud storage services [22] in the environment where ownership changes dynamically. It is achieved by exploiting a group key management mechanism in each ownership group. As compared to the previous deduplication schemes over encrypted data, the proposed scheme has the following advantages in terms of security and efficiency.

First, dynamic ownership management guarantees the backward and forward secrecy of deduplicated data upon any ownership change. As opposed to the previous schemes, the data encryption key is updated and selectively distributed to valid owners upon any ownership change of the data through a stateless group key distribution mechanism using a binary tree. The ownership and key management for each user can be conducted by the semi-trusted cloud server deployed in the system. Thus, the proposed scheme delegates the most laborious tasks of ownership management to the cloud server without leaking any confidential information to it, rather than to the users. Second, the proposed scheme ensures security in the setting of PoW by introducing a re-encryption mechanism that uses an additional group key for dynamic ownership group. Thus, although the encryption key (that is the hash value of the file) is revealed in the setting of PoW, the privacy of the outsourced data is still preserved against outside adversaries, while deduplication over encrypted data is still enabled and data integrity against poison attacks is guaranteed.

1. Convergent encryption exploits a block cipher as an encryption primitive.

2. A multimedia streaming service based on a pay-as-you-go policy in the cloud is a good example of services that have these backward and forward security requirements.

1.2 Organization

The rest of the paper is organized as follows. In Section 3, the system architecture and security requirements are described. In Section 4, the cryptographic background is provided and the general framework of deduplication over encrypted data is defined. In Section 5, we propose our scheme's construction. We analyze the efficiency and security of the proposed scheme in Section 6 and 7, respectively. In Section 8, we conclude the paper.

2 RELATED WORK

Deduplication techniques can be categorized into two different approaches: deduplication over unencrypted data and deduplication over encrypted data. In the former approach, most of the existing schemes have been proposed in order to perform a PoW process in an efficient and robust manner, since the hash of the file, which is treated as a "proof" for the entire file, is vulnerable to being leaked to outside adversaries because of its relatively small size. Whereas, in the latter approach, data privacy is the primary security requirement to protect against not only outside adversaries but also inside the cloud server. Thus, most of the schemes have been proposed to provide data encryption, while still benefiting from a deduplication technique, by enabling data owners to share the encryption keys in the presence of the inside and outside adversaries. Since encrypted data are given to a user, data access control can be additionally implemented by selective key distribution after the PoW process. However, not much work has yet been done to address dynamic ownership management and its related security problem.

2.1 Deduplication over Unencrypted Data

Harnick et al. [11] demonstrated how data deduplication technique can be used as a side channel that reveals information to malicious users about the contents of files of other users. On the basis of Harnick et al.'s study, Halevi et al. [21] also introduced a similar attack scenario on cloud storage that uses deduplication across multiple users. Specifically, when an attacker temporarily compromises a server and obtains the hash values for data in the cloud storage, he is able to download all these data. This is because only a small piece of information about the data, namely, its hash value, serves as not only an index of the data to locate information of the data among a huge number of files, but also a "proof" that anyone who knows the hash value owns the corresponding data. Therefore, any users who can obtain the short hash value for specific data are able to access all the data stored in the cloud storage.

Harnik et al. [11] proposed a randomized threshold to avoid an attack on cloud storage services that use server-side data deduplication by stopping data deduplication. However, their method did not employ client-side data possession proofs to prevent hash manipulation attacks. Mulazzani et al. [22] demonstrated the hash manipulation attack and conducted a practical evaluation of such an attack in Dropbox [1], which is one of the biggest cloud storage providers. Specifically, the authors showed that spoofing the hash value of a file chunk added to the local Dropbox folder allows a malicious user to access files of other Dropbox users, given that the SHA-256 hash values of the file's chunks are known to the attacker.

To overcome these attacks, Halevi et al. [21] introduced and formalized the notion of proof-of-ownership (PoW), where a user proves to a server that he holds a file using Merkle trees, rather than only a short hash value for it. Specifically, Halevi et al.'s scheme encodes a file using an erasure code that is resilient to the erasure of up to α faction of the bits, and then, builds a Merkle tree over the encoded file. Then, a challenge-response protocol between the server and the client verifies the ownership. PoW is closely related to proof of retrievability [23] and proof of data possession [24]. However, proof of retrievability and data possession often use a pre-processing step that cannot be used in the data deduplication procedure.

Despite their significant benefits in terms of saving resources, these deduplication schemes may cause another security vulnerability and reveal users' private data, in particular, when partial information of users' data has already been leaked. Additionally, all of the above deduplication schemes allow the cloud server to store the data in plaintext form and send plain data to users on receipt of request messages after the PoW procedure. Thus, the cloud server should be fully trusted by all the users in the systems, which constitutes a the significant security threat in the pragmatic cloud storage services where the cloud server may learn every customer's private information and maliciously exploit it.

2.2 Deduplication over Encrypted Data

In order to preserve data privacy against inside cloud server as well as outside adversaries, users may want their data encrypted. However, conventional encryption under different users' keys makes cross-user deduplication impossible, since the cloud server would always see different ciphertexts, even if the data are the same, regardless of whether the encryption algorithm is deterministic.

Convergent encryption, introduced by Douceur et al. [15], is a promising solution to this problem. In convergent encryption, a data owner derives an encryption key $K \leftarrow H(M)$, where M is data or a file to be encrypted and H is a cryptographic hash function. Then, he computes the ciphertext $C \leftarrow E(K, M)$ via a block cipher E, deletes M, and keeps only K after uploading C to the cloud storage. If another user encrypts the same message, the same ciphertext C is produced since encryption is deterministic. Thus, on receipt of C from other users after the initial upload, the server does not store the file but instead updates meta-data to indicate it has an additional owner. If any legitimate owners request and download C later, they can decrypt it with K.

However, convergent encryption suffers from the following security flaw. Suppose a user u_a has data M_a and a user u_b has other data $M_b \neq M_a$. u_a uploads maliciously-generated ciphertext $C_a \leftarrow E(H(M_b), M_a)$, and its tag (or, index) $\mathcal{T}(C_a) \leftarrow H(E(H(M_b), M_b))$. Then, when u_b tries to upload $C_b \leftarrow E(H(M_b), M_b)$ and its tag, the server sees a tag match $\mathcal{T}(C_a) = \mathcal{T}(C_b)$. Thus, the server deletes C_b and keeps only C_a. Later, when u_b downloads it, the decryption constitutes M_a, not M_b, meaning the integrity of the data has been compromised. This is referred to as the tag consistency problem [20]. Xu et al. [19] also introduced a similar data integrity attack in the cloud storage service, called a poison attack.

In order to solve this problem, Bellare et al. [20] introduced a message-locked encryption (MLE) concept and its security notion, and proposed randomized convergent encryption as one implementation of MLE. In randomized convergent encryption, an initial uploader encrypts a message and generates $C_1 \leftarrow E(L, M)$, where L is a randomly chosen key, and then encrypts the message encryption key L and generate $C_2 \leftarrow L \oplus K$, where K is a key-encrypting key (KEK) that is derived from the message ($K \leftarrow H(M)$). Then, the message tag T is generated from the KEK, not from the ciphertext ($T \leftarrow H(P, K)$, where P is a set of public parameters). When any legitimate owner receives C_1, C_2, T from the server later, he computes $L \leftarrow C_2 \oplus K$, decrypts C_1 with L, and obtains M. Then, he generates a tag $T' \leftarrow H(P, H(P, M))$ and checks whether $T' = T$. If $T' = T$, he accepts it; else, rejects it, since the data is compromised. In the scheme, C_2 is used to distribute the message encryption key, where K is used as a group KEK shared among owners of the same data. Since a tag is generated from the KEK, not from the ciphertext, even different ciphertexts encrypted under the different keys of each owner can be deduplicated provided that the plaintext is the same. Xu et al. [19] also proposed a leakage-resilient deduplication scheme to resolve the data integrity problem. This scheme also enables the data owner to encrypt data with a randomly selected key. Then, the data encryption key is encrypted under a KEK derived from the data and distributed to the other data owners after the PoW process. If a legitimate owner receives a ciphertext, he can check the integrity of the data by decrypting the data encryption key with the same KEK.

Convergent encryption is insecure in the setting of PoW, where the hash value of the file (that is, a deterministic encryption key) may be leaked [19],[21]. Unfortunately, this is also the case in MLE [20] and Xu et al.'s schemes [19]. Since the hash value of the file is used as the KEK in both schemes, if the KEK is revealed, adversaries who obtain it are able to decrypt the key encryption message and obtain the encryption key, even if the encryption key is not deterministic. Another drawback in both schemes is the lack of dynamic ownership management among the data owners. For example, suppose a group of users share data in the cloud storage. Some users may request data deletion or modification in the storage. Then, they should be prevented from accessing the original data after this time instance (forward secrecy). Likewise, when a user subsequently uploads the data, access right to the previous data should not be given to him before that time instance (backward secrecy). However, in both schemes, this unauthorized data access cannot be controlled, since the data encryption key cannot be updated at all after its initial selection and distribution by an initial uploader.

Recently, Li et al. [25] proposed a convergent key management scheme in which users distribute the convergent key shares across multiple servers by exploiting the Ramp secret sharing scheme [26]. Li et al. [27] also proposed an authorized deduplication scheme in which differential privileges of users, as well as the data, are considered in the deduplication procedure in a hybrid cloud environment. Jin et al. [31] proposed an anonymous deduplication scheme over encrypted data that exploits a proxy re-encryption algorithm. Bellare et al. [28] proposed a server-aided MLE which is secure against brute-force attack, which was recently extended to interactive MLE [29] to provide privacy for messages that are both correlated and dependent on the public system parameters. However, these schemes do not handle the dynamic ownership management issues involved in secure deduplication for shared outsourced data.

Shin et al. [30] proposed a deduplication scheme over encrypted data that uses predicate encryption. This approach allows deduplication only of files that belong to the same user, which severely reduces the effect of deduplication. Thus, in this paper, we focus on deduplication across different users such that identical files from different users are detected and deduplicated safely to provide more storage savings.

3 Data Deduplication Architecture

In this section, we describe the data deduplication architecture and define the security model. According to the granularity of deduplication, deduplication schemes are categorized into (coarse-grained) file-level or (fine-grained) block-level schemes. Since block-level deduplication can easily be deduced from file-level deduplication, we consider only file-level

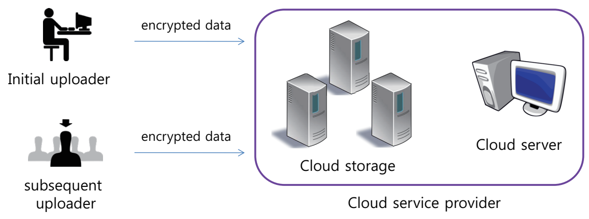

Fig. 1. Architecture of a data deduplication system

deduplication for simplicity's sake. Thus, a data copy refers to a whole file in this paper.

3.1 System Description and Assumptions

Fig. 1 shows the architecture of the data deduplication system, which consists of the following entities.

1) Data owner: This is a client who owns data, and wishes to upload it into the cloud storage to save costs. A data owner encrypts the data and outsources it to the cloud storage with its index information, that is, a tag. If a data owner uploads data that do not already exist in the cloud storage, he is called an initial uploader; if the data already exist, called a subsequent uploader since this implies that other owners may have uploaded the same data previously, he is called a subsequent uploader. Hereafter, we refer to a set of data owners who share the same data in the cloud storage as an ownership group.
2) Cloud service provider: This is an entity that provides cloud storage services. It consists of a cloud server and cloud storage. The cloud server deduplicates the outsourced data from users if necessary and stores the deduplicated data in the cloud storage. The cloud server maintains ownership lists for stored data, which are composed of a tag for the stored data and the identities of its owners. The cloud server controls access to the stored data based on the ownership lists and manages (e.g., issues, revokes, and updates) group keys for each ownership group as a group key authority. The cloud server is assumed to be honest-but-curious. That is, it will honestly execute the assigned tasks in the system; however, it would like to learn as much information about the encrypted contents as possible. Thus, it should be deterred from accessing the plaintext of the encrypted data even if it is honest.

3.2 Threat Model and Security Requirements

1) Data privacy: Unauthorized users who cannot prove ownerships should not be able to decrypt the ciphertext stored in the cloud storage. Additionally, the cloud server is no longer fully trusted in the system. Thus, unauthorized access from the cloud server to the plaintext of the encrypted data in the cloud storage should be prevented.
2) Data integrity: The deduplication algorithm should guarantee tag consistency against any poison attacks. That is, the deduplication algorithm should allow the valid owners to verify that the data downloaded from the cloud storage have not been altered.
3) Backward and forward secrecy[3]: In the context of deduplication, backward secrecy means that any user should be prevented from accessing the plaintext of the outsourced data before uploading the data. Conversely, forward secrecy means that any user who deletes or modifies the data in the cloud storage should be prevented from accessing the outsourced data after its deletion or modification.
4) Collusion resistance: Unauthorized users who do not have valid ownerships of data in the cloud storage should not be able to decrypt them even if they collude.

4 Preliminaries and Definition

4.1 Notations

In this paper, $x \overset{\$}{\leftarrow} S$ denotes the operation of selecting an element x at random and uniformly from a finite set S and assigning it to x. For an algorithm $\mathcal{A}$, $y \leftarrow \mathcal{A}(x_1, \ldots)$ denotes running $\mathcal{A}$ on inputs $x_1, \ldots$ and assigning the output to the variable y. 1^λ denotes a string of λ ones, if $\lambda \in \mathbb{N}$, which is the security parameter[4]. For two bit-strings a and b, we denote by $a||b$ their concatenation.

Let $\mathcal{U} = \{u_1, \cdots, u_n\}$ be the universe of users. Let ID_t be the identity of a user u_t. Let $G_i \subset \mathcal{U}$ be a set of users that owns the data M_i, which is referred to as an ownership group. Let $L_i = \langle T_i, G_i \rangle$ be an ownership list for M_i, maintained by the cloud server, which consists of a tag T_i and G_i for M_i. Let K_{G_i} be the ownership group key that is shared among the valid owners in G_i.

4.2 Definitions

In this section, we define a secure deduplication framework for encrypted data with ownership management capability. The scheme consists of the following algorithms:

1) $KEK \overset{\$}{\leftarrow} \mathsf{KEKGen}(U)$: The KEK generation algorithm takes a set of users U as input, and outputs

3. In secure group communication, backward secrecy implies that when a member newly joins a multicast group, he should be prevented from learning group communications exchanged before he joins the group. Forward secrecy implies when a member leaves a multicast group, he should be prevented from learning group communications exchanged after he leaves the group [35].

4. To be consistent with the standard convention in algorithms, where the running time of an algorithm is measured as a function of the length of its input, we will provide the adversary and the honest parties with the security parameter in unary as 1^λ.

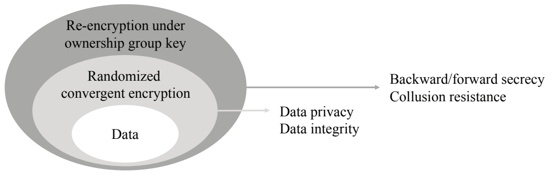

Fig. 2. Scheme overview and corresponding security

KEKs for each user in U for secure ownership group key distribution.

2) $C \overset{\$}{\leftarrow}$ Encrypt$(M, 1^\lambda)$: The encryption algorithm is a randomized algorithm that takes as input data M and a security parameter λ, and outputs a ciphertext C of the data. C consists of the encrypted message and its tag information for indexing.
3) $C' \overset{\$}{\leftarrow}$ ReEncrypt(C, G): The re-encryption algorithm is a randomized algorithm that takes a ciphertext C and an ownership group G, and outputs a re-encrypted ciphertext C'. Specifically, it outputs a re-encrypted ciphertext such that only valid owners in G can decrypt the message.
4) $M \leftarrow$ Decrypt(C', K, PK): The decryption algorithm is a deterministic algorithm that takes as input C', message encryption key K, and a set of KEKs PK for encrypting an ownership group key GK, and outputs a message M, iff K is derived from M and GK is not revoked for the ownership group G (that is, the decryptor is in G) for M.

5 Proposed Deduplication Scheme

In this section, we propose a secure deduplication scheme for encrypted data that has dynamic ownership management capability. The proposed scheme is constructed based partially on a randomized convergent encryption scheme [20] in order to randomize the encrypted data, which renders the proposed scheme secure against the chosen-plaintext attack while still allowing deduplication over the data. The proposed scheme is further integrated into the re-encryption protocol for owner revocation. The owner revocation is executed by re-encrypting the outsourced ciphertext and selectively distributing the re-encryption key to valid (that is, not revoked) owners by the cloud server. Fig. 2 shows the overview of the proposed scheme and its corresponding security goals.

To handle dynamic ownership management, the cloud server must obtain the ownership list for each data, since otherwise revocation cannot take effect. This setting where the cloud server knows the ownership list does not violate the security requirements, because it is allowed only to re-encrypt the ciphertexts and can by no means obtain any information about the data encryption key of users.

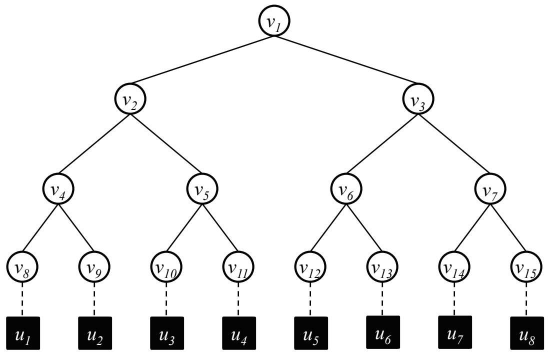

Fig. 3. KEK tree for ownership group key distribution

5.1 Scheme Construction

Let $E_K(M)$ be a symmetric encryption of a message M under a key K. The simplest implementation is to make $E_K : \{0,1\}^k \rightarrow \{0,1\}^k$ a block cipher, where k is the length of the key K. We additionally employ a cryptographic hash function $H : \{0,1\}^* \rightarrow \{0,1\}^*$ to generate an encryption key and a tag from a message.

5.1.1 Key Generation

The cloud server runs KEKGen(U) and generates KEKs for users in U. First, the cloud server sets a binary KEK tree for the universe of users $\mathcal{U}$, as in Fig. 3, which will be used to distribute the ownership group keys to users in $U \subseteq \mathcal{U}$. In the tree, each node v_j holds a KEK, denoted by KEK_j. A user is represented by a leaf, and each user maintains the KEKs on the path nodes from its leaf to the root. These are called *path keys*. For instance, in Fig. 3, u_2 stores KEK_9, KEK_4, KEK_2, and KEK_1 as its path keys PK_2. For $u_t \in \mathcal{U}$, PK_t denotes a set of the path keys of u_t. The KEK tree is constructed by the cloud server as follows:

1) Every member in $\mathcal{U}$ is assigned to a leaf node of the tree. Random keys are generated and assigned to each leaf node and internal node.
2) Each member $u_t \in U$ receives securely the path keys PK_t from its leaf node to the root node of the tree.

Then, the path keys are used as KEKs to encrypt the ownership group keys by the cloud server in the data re-encryption phase. The key assignments in this method are conducted randomly and independently of each other.

5.1.2 Data Encryption

Without loss of generality, we suppose a data owner u_t wants to upload his data M_i to the cloud storage. u_t encrypts the data by running the Encrypt$(M_i, 1^\lambda)$ algorithm.

The algorithm chooses a random data encryption key $L \overset{\$}{\leftarrow} \{0,1\}^{k(\lambda)}$, where $k(\lambda)$ is an algorithm that determines the size of the encryption key under the security parameter. It also computes a key $K_i \leftarrow H(M_i)$ from M_i, which will be used as a KEK for encrypting a message encryption key L, and a tag $T_i \leftarrow H(K_i)$, which is an index information for the data. Then, the algorithm encrypts the data and the encryption key as $C_i^1 \leftarrow E_L(M_i)$ and $C_i^2 \leftarrow L \oplus K_i$, and constructs the ciphertext $C_i = C_i^1||C_i^2$.

After the construction of C_i, the data owner u_t sends $upload||T_i||C_i||ID_t$ to the cloud storage[5]. Then, the owner deletes M_i and retains only K_i for storage saving. On its receipt, the cloud server inserts ID_t into G_i, creates $L_i = \langle T_i, G_i \rangle$, and stores C_i in the cloud storage, if u_t is the first uploader for M_i. If L_i already exists (which means u_t is a subsequent uploader), then it inserts only ID_t into G_i without storing C_i.

5.1.3 Data Re-encryption

Before distributing the ciphertext C_i, the cloud server re-encrypts it by running ReEncrypt(C_i, G_i) using the ownership group information for the ciphertext. The re-encryption algorithm enforces access control of dynamically changing owners to the outsourced data.

The algorithm progresses as follows:

1) For G_i, choose a random ownership group key GK_i. Then, re-encrypt C_i^1 and generate $C_i^{1'} = E_{GK_i}(C_i^1)$.
2) Select root nodes of the minimum cover sets in the KEK tree that can cover all of the leaf nodes associated with users in G_i. We denote by $KEK(G_i)$ a set of KEKs that such root nodes of subtrees for G_i hold. For example, if $G_i = \{u_1, u_2, u_3, u_4, u_7, u_8\}$ in Fig. 3, then $KEK(G_i) = \{KEK_2, KEK_7\}$, because v_2 and v_7 are the root nodes of the minimum cover sets that can cover all of the members in G_i. It follows that this collection covers all users in G_i and only them, and any user $u \notin G_i$ can by no means know any KEK in $KEK(G_i)$.
3) Generates
$$C_i^3 = \{E_K(GK_i)\}_{K \in KEK(G_i)}.$$
This encryption is employed as the method for delivering the ownership group keys to valid owners.

On receiving any data request query $T_i||ID_j$ from a user u_j, the cloud server looks up L_i and responds with $T_i||C_i'$ to the user, where $C_i' = C_i^{1'}||C_i^2||C_i^3$ if $ID_j \in G_i$; otherwise, it does nothing. The former indicates the case where the user has uploaded the data and has not been revoked, while the latter indicates the case where the user has not uploaded the data or has been revoked at this moment.

It is important to note that the ownership group key distribution protocol through C_i^3 is a stateless approach. Thus, even if users cannot update their key states constantly (e.g., in case of offline), they are able to decrypt the group key from it at any time they receive it when they become online, provided that they are not revoked from the ownership groups.

5.1.4 Data Decryption

When a user u_t receives a ciphertext C_i' from the cloud server, he can decrypt the message by running Decrypt(C_i', K_i, PK_t), if $u_t \in G_i$. The data decryption phase consists of ownership group key decryption followed by the message decryption.

Ownership Group Key Decrypt. When a user sends a data request query and receives $T_i||C_i'$ from the cloud server, he first parses it as T_i, $C_i^{1'}$, C_i^2, C_i^3, and obtains the ownership group key from C_i^3. If the user u_t has valid ownership (that is, $u_t \in G_i$ at this time instance), he can decrypt the ownership group key GK_i using a KEK $\in KEK(G_i) \cap PK_t$ as
$$GK_i = D_{KEK \in (KEK(G_i) \cap PK_t)}(C_i^3).$$

The user u_t may belong to at most one subset rooted by only one such KEK in $KEK(G_i)$. Thus, there can be only one such KEK.

The key-indistinguishability property follows from the fact that no $u \notin G_i$ is contained in any of the subsets the root node of which is holding any KEK in $KEK(G_i)$. This means that, for every KEK in $KEK(G_i)$, the KEK is indistinguishable from a random key, given all the information of all users not in G_i [32]. Thus, any user $u \notin G_i$ can by no means decrypt GK_i, even if he colludes with other users $u' \notin G_i$, which makes the proposed scheme secure against such a collusion attack as we will analyze it in Section 7.4.

Message Decrypt. After that, the user u_t decrypts the ciphertext and obtains the message as follows.
$$C_i^1 \leftarrow D_{GK_i}(C_i^{1'}),\ L \leftarrow C_i^2 \oplus K_i,$$
$$M_i \leftarrow D_L(C_i^1),\ T_i' \leftarrow H(M_i).$$

If $T_i' \neq T_i$, which represents tag inconsistency, the user drops the message, since it may be modified under a poison attack; else, the user accepts M_i as the original data he outsourced. For example, if a malicious user uploads a false message M' with a tag $H(M)$ where $M' \neq M$, a subsequent user who owns M could detect the pollution of original data M by decrypting the ciphertext, obtaining M', and checking $H(M') \neq H(M)$. Then, the subsequent user would report the data inconsistency to the cloud server, which may help the cloud server to find and revoke

5. In server-side deduplication approaches, the data owner may send only T_i, and the cloud server requests the owner to upload the encrypted data only when there are no data indexed by T_i. This approach can save the network bandwidth; however, it can be used as a side channel that reveals information about the contents of files of other users. This may violate the privacy of other users [11]. Thus, in the proposed scheme, we assume that the data owner sends C_i as well as T_i in order to preserve privacy.

the malicious user, and delete the polluted data from the cloud storage[6].

5.2 Key Update

When subsequent users upload data which is the same as the previously uploaded by the initial uploader, the corresponding ciphertext should be re-encrypted to prevent subsequent users from accessing the previous encrypted data in order to provide backward secrecy. In contrast, when users who have valid ownerships request the cloud server to delete or modify the data in the cloud storage, they should be revoked from the ownership list and deterred from accessing the data after data deletion or modification in order to provide forward secrecy.

Subsequent Upload. We suppose a user u_s wants to upload data M_i to the cloud storage, and its corresponding ownership list $L_i = \langle T_i, G_i \rangle$ and ciphertext $C_i = C_i^{1\prime}||C_i^2$ already exist in the cloud storage (C_i might be encrypted and uploaded by the initial uploader, and re-encrypted by the cloud server such that $C_i^{1\prime} = E_{GK_i}(C_i^1)$). Then, the user u_s encrypts the data by running the Encrypt(M_i, 1^λ) algorithm and generates ciphertext, say C_i''. With overwhelming probability, it holds that $C_i'' \neq C_i$ since the encryption key L is randomly selected from $\{0,1\}^{k(\lambda)}$ by different users. After the construction of C_i'', the data owner u_s sends $upload||T_i'||C_i''||ID_s$ to the cloud storage[7]. Then, the key update and re-encryption processes progress as

1) If $T_i' = T_i$, the cloud server puts ID_s into G_i.
2) The cloud server decrypts the ciphertext component $C_i^{1\prime} = E_{GK_i}(C_i^1)$ in C_i with the current ownership group key GK_i. Then it selects a random ownership group key $GK_i'(\neq GK_i)$ and runs the ReEncrypt(C_i, G_i) algorithm described in Section 5.1.3 with the updated ownership group information G_i and GK_i' to guarantee backward secrecy, which updates the ciphertext component as $C_i^{1\prime} : E_{GK_i}(C_i^1) \rightarrow E_{GK_i'}(C_i^1)$.

If there is no T_i in the cloud storage such that $T_i' = T_i$, since this implies the first upload, the cloud server creates a new ownership list for the data, inserts ID_s into the newly generated ownership group, and stores the uploaded data in the cloud storage following the same procedures described in Section 5.1.2.

Data Deletion. When a user u_s wants to delete data M_i from the cloud storage, the user sends the data deletion request messages with $delete||T_i||ID_s$ to the cloud server. Then, the cloud server performs the following procedures.

1) If $ID_s \in G_i$, it deletes ID_s from G_i. Then, it selects a random ownership group key and runs the ReEncrypt(C_i, G_i) algorithm described in Section 5.1.3 with the updated ownership group information G_i to guarantee forward secrecy.
2) Else, it does nothing.

Data Modification. When a user u_s wants to modify the data M_i to M_j, the user encrypts the data and constructs the ciphertext C_j and its corresponding tag T_j by running the Encrypt(M_j, 1^λ) algorithm described in Section 5.1.2. Then, the user sends a data modification request message with $modify||T_i||T_j||C_j||ID_s$ to the cloud server. Then, the cloud server performs the data deletion procedure, followed by data upload procedure, as follows.

- Data Deletion(1–2):

1) If $ID_s \in G_i$, it deletes ID_s from G_i. Then, it selects a random ownership group key and runs the ReEncrypt(C_i, G_i) algorithm described in Section 5.1.3 with the updated ownership group information G_i for guaranteeing forward secrecy.
2) Else, it does nothing and stops running the algorithm.

- Data Upload(3–4):

3) If there exists $L_j = \langle T_j, G_j \rangle$ for the tag T_j in the cloud storage, it performs the subsequent upload procedures described above.
4) Else, if there does not exist L_j in the cloud storage, it performs the initial upload procedures described in Section 5.1.2.

When multiple users upload or delete the same file at the same time, they are handled in a batch way. Specifically, the ownership list for the file is updated based on the ownership changes, and the corresponding ownership group key and ciphertext are updated once. Then, they are securely delivered following the proposed algorithms. We note that this can be handled straightforwardly without any security degradation.

5.3 Comparison

Table 1 shows the comparison results of the secure data deduplication schemes, that is convergent encryption (CE) [15], leakage-resilient (LR) deduplication [19], and randomized convergent encryption (RCE) [20] in terms of the data deduplication over

6. Even if the proposed scheme can detect any data modification or loss by a malicious user or CSP, it cannot recover the original data under the data loss attack (e.g., by malicious CSP or Byzantine failure) because all of the redundant data would be deduplicated. One of the approaches for deduplicated data recovery is to adopt information dispersal techniques [36],[37],[38] that transform data into multiple shares with some redundancy and disperse the shares across multiple CSPs, which is out of scope in this paper.

7. The subsequent uploader sends C_i'' to prevent the side-channel attack [11] as in the initial upload; however, if communication channel is secure in the presence of eavesdroppers, C_i'' does not need to be uploaded, which will reduce the communication cost as in the client-side deduplication.

TABLE 1
Comparison of secure deduplication schemes

Scheme	Encrypted deduplication	Tag consistency	Ownership management
CE [15]	yes	no	no
LR [19]	yes	yes	no
RCE [20]	yes	yes	no
Proposed	yes	yes	yes

encrypted data, tag consistency, and dynamic ownership management.

Since all the schemes allow data owners to encrypt their data and enable deduplication over them, they can guarantee the data confidentiality or privacy against the cloud server and unauthorized outside adversaries. With regard to data integrity, convergent encryption cannot guarantee the integrity of deduplicated data in the face of a poison attack, whereas the other schemes preserve it by adopting an additional mechanism that enables data owners to check the tag consistency of the received data.

In the proposed scheme, upon every membership change in the ownership list (e.g., subsequently uploading the same data, or modifying/deleting the existing data), access to the corresponding data is permitted to owners only for the time windows during which the owners maintain valid ownership of the data by re-encrypting it using an updated ownership group key and selectively distributing it. This resolves the dynamic ownership management problem as opposed to the other schemes. The rekeying in the proposed scheme can be done immediately upon any ownership change. This enhances the security of the outsourced data in terms of backward/forward secrecy by reducing the windows of vulnerability. A more rigorous security analysis is given in Section 7.

6 Scheme Analysis

In this section, we analyze the efficiency of the proposed scheme and compare it with the previous deduplication schemes over encrypted data in terms of both the theoretical and practical aspects. The efficiency of the proposed scheme is demonstrated in the network simulation in terms of communication cost. We also discuss its computation cost when implemented with specific parameters.

6.1 Efficiency

The comparative results for the theoretical efficiency of the schemes are summarized in Table 2. In Table 2, the analysis results of each scheme in terms of the communication and storage overhead are shown. For communication overhead, "upload message size" represents the communication cost required for the data outsourcing process; "download message size" represents the communication cost required for ciphertext downloading and tag checking processes, and "rekeying message size" represents the communication cost required for rekeying the data encryption key. For storage overhead, "key size" and "tag size" represent the size of the keys and tag information that each owner needs to store, respectively.

The notations used in the table are as follows.

C_M	Size of a data or file
C_C	Size of an encrypted data (= output length of $E(\cdot)$)
C_K	Size of a key (= output length of $k(\lambda)$ on input 1^λ)
C_T	Size of a tag
C_{ID}	Size of an identity of a user ($\geq \log n$)
C_r	Size of a node value in Merkle hash tree
C_{PoW}	Size of exchanged messages for PoW on inputs the file size and 1^λ ($= u\log C_M$, where u is the smallest integer such that $(1-\alpha)^u < \varepsilon$ for some constant fraction $\alpha > 0$) [21]
n	Number of users in the system
m	Number of owners in an ownership list for a file

For the upload and download message sizes, the proposed scheme is the same as the basic RCE [20] scheme. In LR [19], the communication overhead for verifying PoW is additionally included in the download message. In the scheme, the PoW verification and tag checking processes are done during the data upload phase by subsequent owners. However, they can be executed during the data download phase without loss of functionality and efficiency. Thus, we suppose they are executed during the download phase as in RCE [20] and the proposed scheme for the sake of fair comparison.

With regard to the rekeying message size, only the proposed scheme supports key updates upon ownership changes for data. In the proposed scheme, the rekeying message size (i.e., size of C_i^3) would be $(n-m)\log\frac{n}{n-m}C_k$. This additional message plays an important role in enhancing the backward and forward secrecy, and enforces fine-grained user access control to the outsourced data in contrast to the other schemes. Whereas, in CE [15], the encryption key is determined by the message itself; in LR [19] and RCE [20], it is selected by the initial uploader and never updated during the lifetime of the data in the system. Thus, even if the other schemes do not need the additional rekeying messages, they cannot guarantee the data privacy during the windows of vulnerability in the practical cloud environment where the ownership changes dynamically as time elapses.

With regard to storage overhead, in the LR scheme, each owner stores the leaf node values of the Merkle tree for PoW in addition to the data encryption key and KEK, the size of which increases in proportion to the data size. In the proposed scheme, each data

TABLE 2
Efficiency comparison

Scheme	Communication overhead			Storage overhead	
	Upload message size	Download message size	Rekeying message size	Key size	Tag size
CE [15]	$C_C + C_T + C_{ID}$	C_C		C_K	C_T
LR [19]	$C_C + 3C_K + C_T + C_r + C_{ID}$	$C_C(+C_{PoW} + C_K + C_T)^\dagger$		$2C_K + C_M \cdot C_r$	C_T
RCE [20]	$C_C + C_K + C_T + C_{ID}$	$C_C + C_K + C_T$		C_K	C_T
Proposed	$C_C + C_K + C_T + C_{ID}$	$C_C + C_K + C_T$	$(n-m)\log\frac{n}{n-m}C_k$	$(\log n + 1)C_K$	C_T

†: communication overhead to check tag consistency

owner stores log$n$ additional KEKs as compared to the original RCE scheme. These KEKs allow the secure and selective distribution of the dynamically updated data encryption key, which supports fine-grained access control on the basis of the valid ownership of each user with a little storage overhead.

6.2 Simulation

In this simulation, we measure the communication cost of the deduplication schemes. We consider the online cloud storage systems connected to the Internet. Almeroth et al. [33] demonstrated the group behavior in the Internet's multicast backbone network (MBone). They showed that the number of users joining a multicast group follows a Poisson distribution with rate λ, and the membership duration time follows an exponential distribution with a mean duration $1/\mu$. Since each owner group of data or files can be seen as an independent network group, where the owners in the group share common outsourced data, we show the simulation results following this probabilistic behavior distribution [33].

We suppose that user join and leave events are independently and identically distributed in each ownership group following Poisson distribution. The ownership duration time for outsourced data (that is, from data upload time to deletion time) is assumed to follow an exponential distribution. We set the inter-upload time between users as 12 hours ($\lambda = 2$) and the average ownership duration time as 10 days ($1/\mu = 10$).

6.3 Implementation

Figs. 4 and 5 show the total communication costs that the cloud server incurs to send on data requests from owners in the secure deduplication schemes that support tag consistency (that is, LR [19], RCE [20], and the proposed scheme) in a single ownership group during 100 days. In this simulation, we suppose the owners send data request messages immediately after ownership changes in the ownership group to measure the communication cost incurred by dynamic ownership changes for a fair comparison with regard to the security perspective. The communication cost, which is measured in bytes, includes those of the

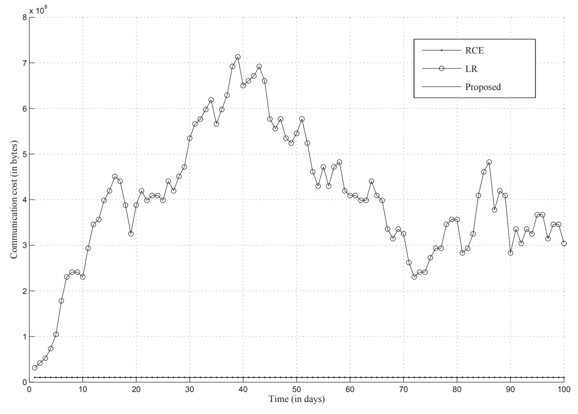

Fig. 4. Communication cost in LR, RCE, and the proposed scheme

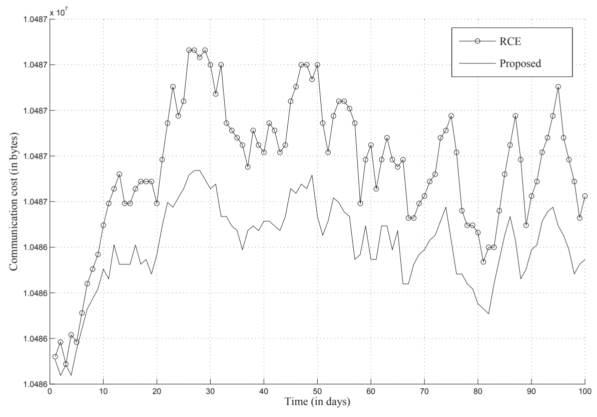

Fig. 5. Communication cost in RCE and the proposed scheme

ciphertext and of rekeying messages for non-revoked owners.

In this simulation, we set $C_M = C_C = 10$MB for a typical multimedia data (e.g., music file), which is one of the most commonly used types of data in the cloud. To achieve a 128-bit security level, we set $C_K = 128$ bits, $C_T = 128$ bits. Fig. 4 shows the communication costs in LR, RCE, and the proposed scheme. In the LR scheme, each owner performs a PoW process on every data request procedure and receives an updated encryption key and ciphertext by unicast. This incurs much more communication overhead than the other schemes. Thus, RCE and the proposed scheme have

TABLE 3
Comparison of computation cost

Operation	Time (ms)	CE [15]		LR [19]		RCE [20]		Proposed scheme	
		upload	download	upload	download	upload	download	upload	download
Hash	0.025	2		5	1	2	2	2	2
Data encrypt	Enc	1		2		1		1	
Data decrypt	Dec		1		1		1		1
Key decrypt	0.129								1
Computation (ms)		$Enc + 0.050$	Dec	$2Enc + 0.125$	$Dec + 0.025$	$Enc + 0.050$	$Dec + 0.050$	$Enc + 0.050$	$Dec + 0.179$

almost the same and relatively negligible communication overhead, as shown in Fig. 4. To provide a detailed comparison of the two schemes, Fig. 5 shows the communication costs of RCE and the proposed scheme. In RCE, we assume that only the rekeying component $C_2 = L \oplus K$ is selectively distributed to valid owners securely by unicast in order to realize a dynamic ownership management similar to that of the proposed scheme, which is the most efficient rekeying scenario in RCE. In this scenario, the communication overhead of the proposed scheme is less than that of RCE by about 300-byte with the same level of ownership management capability[8].

Next, we analyze and measure the computation cost incurred when a data owner encrypts and decrypts data during upload and download phases, respectively. The computation cost is shown in Table 3 in terms of the computation of a cryptographic hash function for key generation, tag generation (the hash function is also used for key encryption/decryption in LR [19]), data encryption/decryption, and key decryption. The comparatively negligible bitwise exclusive-or operations are ignored in the computation analysis results.

For each operation, we include a benchmark timing. Each cryptographic operation was implemented using the Crypto++ library ver. 5.6.2 [34] on a 3.4 GHZ processor PC. The key parameters were selected to provide a 128-bit security level. The implementation uses an MD5 as a cryptographic hash function to generate a 128-bit key and tag, and an AES with Electronic Code Book (ECB) mode as an encryption/decryption function. Data encryption and decryption time, denoted by Enc and Dec, respectively, in Table 3, are measured for different data sizes as shown in Fig. 6, which increase in proportion to the size of data.

On the basis of the encryption and decryption time, we measured the total computation cost for the upload and download of each scheme, as shown in Fig. 7 and Fig. 8, respectively. For the upload procedure, the proposed scheme requires the same computations as the CE and RCE schemes. For the download procedure, the proposed scheme needs one more key decryption operation than does the basic RCE scheme. However, since the symmetric key size is much smaller than the typical data size in the cloud (e.g., document file, or multimedia data), the additional 128-bit key decryption time (i.e., $0.129\ ms$) in the proposed scheme would be relatively negligible as compared to the data decryption time in a pragmatic cloud computing system as depicted in Fig. 8.

8. Each detailed communication cost can be found in the supplementary file for this paper.

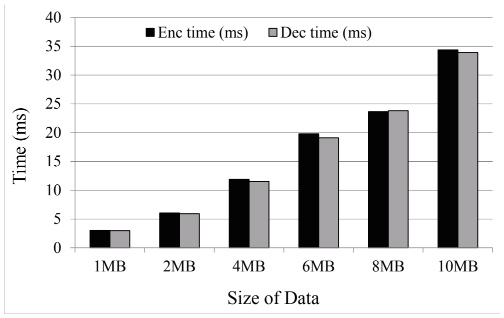

Fig. 6. Encryption and decryption time

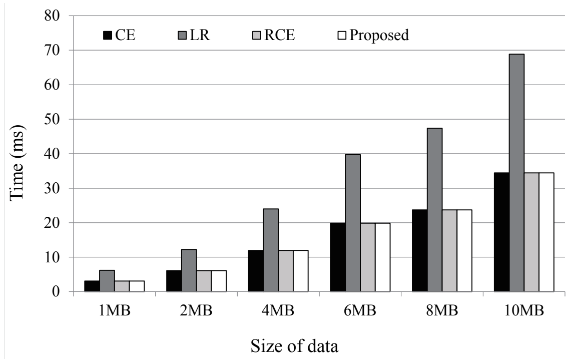

Fig. 7. Computation time for upload

The measured computation time for upload and download is described in Table 4. More experimental results with diverse file size (100KB $\sim$ 1000MB) can be found in the supplementary file for this paper.

7 Security

In this section, we prove the security of the proposed scheme in terms of the security requirements discussed in Section 3.2, that is data privacy, data integrity, backward and forward secrecy, and collusion resistance.

TABLE 4
Upload and down load time (ms)

Data size	CE [15]		LR [19]		RCE [20]		Proposed scheme	
	upload	download	upload	download	upload	download	upload	download
1MB	3.095	3.045	6.215	3.07	3.095	3.095	3.095	3.224
2MB	6.103	6.053	12.231	6.078	6.103	6.103	6.103	6.232
4MB	11.972	11.922	23.969	11.947	11.972	11.972	11.972	12.101
6MB	19.853	19.803	39.731	19.828	19.853	19.853	19.853	19.982
8MB	23.691	23.641	47.407	23.666	23.691	23.691	23.691	23.82
10MB	34.425	34.375	68.875	34.4	34.425	34.425	34.425	34.554

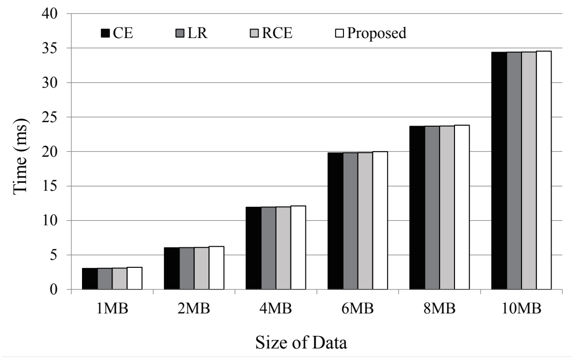

Fig. 8. Computation time for download

7.1 Data Privacy

In our trust model, the cloud server is no longer fully trusted even if honest. Therefore, plain data should be kept secret from the cloud server as well as from unauthorized users who cannot prove ownership.

Data privacy for the outsourced data against unauthorized users who have never owned the data can be trivially guaranteed. Without loss of generality, we suppose an unauthorized user requests data for M_i with a tag T_i and one of the other valid users' identity $ID_t \in G_i$ (if $ID_t \notin G_i$, the cloud server would simply reject the request), and receives the corresponding ciphertext $T_i||C_i^{1'}||C_i^2||C_i^3$ where $C_i^{1'} = E_{GK_i}(E_L(M_i))$ and $C_i^2 = L \oplus K_i$. If the user has never owned the data M_i, it is computationally infeasible to guess K_i and obtain the data encryption key L because of the cryptographic hash function. On the other hand, if the user has once owned the data but does not have valid ownership at the request time instance, he may have knowledge of K_i but can by no means obtain the corresponding GK_i from C_i^3, since it is encrypted using KEKs that the user cannot know. Thus, the unauthorized user cannot obtain M_i from $C_i^{1'}$ without GK_i, even if he may be able to obtain L using K_i.

Another attack scenario can be launched by the cloud server. In the cloud server's point of view, the information it can learn is the set of ciphertext $C_i = C_i^1||C_i^2$ with the corresponding L_i. Although the cloud server determines the ownership group key GK_i, it is computationally infeasible for the cloud server to guess K_i, since it does not have knowledge of M_i, like the previous unauthorized attacker who has never owned the data. Without K_i, it cannot decrypt $C_i^{1'}$ and obtain M_i. Therefore, data privacy against the honest-but-curious cloud server and unauthorized users is guaranteed.

7.2 Data Integrity

In the deduplication scheme, data integrity may be threatened by a poison attack on tag consistency. Without loss of generality, we suppose an attacker and another user u have the same data M. The attacker maliciously generates ciphertext C' from $M'(\neq M)$, and uploads it with a tag T generated from M initially.

In the proposed scheme, the poison attack on tag consistency is easily detected, as in the basic RCE scheme. When the user u subsequently requests the data and receives the corresponding ciphertext $C^1||C^2||C^3$ with T, he can obtain the ownership group key GK and data encryption key L from C^3 and C^2, respectively if the user has valid ownership of the data. Then, the user decrypts C^1 and obtains the data M' using the two encryption keys GK and L, generates the tag T' from M', and checks whether $T' = T$. If these tags are not consistent, the user drops the message, since it implies the data may have been modified during the previous outsourcing procedure. Therefore, the proposed scheme guarantees data integrity against a poison attack on tag consistency.

7.3 Backward and Forward Secrecy

When a user holds data that has been already uploaded previously at some time instance and tries to upload them into the cloud storage, the corresponding ownership group key is updated independently and randomly, say from GK to GK', and delivered securely to the valid owners of the data (including the user) immediately. In addition, the ciphertext component $C^{1'} = E_{GK}(C^1)$, which was encrypted with GK, is re-encrypted by the cloud server with the updated ownership group key GK' at the same time as $C^{1''} = E_{GK'}(C^1)$. Even if the user has stored the previous ciphertext before he holds the data, he cannot decrypt the previous ciphertext. This is because, even if he is able to obtain encryption keys

L and GK' from the current ciphertext, they are of no use for recovering the desired component C^1 for the previous ciphertext, since it is re-encrypted with a previous GK. Therefore, the backward secrecy of the outsourced data is guaranteed in the proposed scheme.

On the other hand, when a user deletes or modifies the data at some time instance, the corresponding ownership group key is also updated independently and randomly, say from GK to GK', and delivered securely to the valid ownership group members (excluding the user) immediately. Then, the ciphertext component $C^{1'} = E_{GK}(C^1)$ that was encrypted with GK is re-encrypted by the cloud server with a new ownership group key GK' at the same time as $C^{1''} = E_{GK'}(C^1)$. Then, the user cannot decrypt the current ciphertext after his revocation, since he can by no means obtain GK'. Even if the user has recovered the data encryption key L before he was revoked from the ownership group and stored it, it would be of no use for obtaining the desired data from C^1 in the subsequent ciphertext, since it is re-encrypted with a new random GK'. Therefore, the forward secrecy of the outsourced data is also guaranteed in the proposed scheme.

7.4 Collusion Resistance

To provide collusion resistance, unauthorized users who have not valid ownerships of cloud data should not be able to decrypt them even if they collude. In the proposed scheme, in order to decrypt the ciphertext and obtain the plain data, users should have knowledge of both the data encryption key L and the ownership group key GK. Even if some unauthorized users may be able to obtain the data encryption key[9], it is impossible to have the ownership group key GK. This is because the KEK assignment for ownership group key distribution in the binary KEK tree is information theoretic, that is KEKs are assigned randomly and independently of each other. Whenever any ownership change occurs in an ownership group, the ownership group key is rekeyed immediately and the data are re-encrypted using the updated group key. Even if the unauthorized users collude with each other, they cannot obtain the current ownership group key, since none of the KEKs in their path keys in the KEK tree is used to encrypt and distribute the ownership group key. Therefore, the proposed scheme is secure against a collusion attack of the unauthorized users.

8 Conclusion

Dynamic ownership management is an important and challenging issue in secure deduplication over encrypted data in cloud storage. In this study, we proposed a novel secure data deduplication scheme to enhance a fine-grained ownership management by exploiting the characteristic of the cloud data management system. The proposed scheme features a re-encryption technique that enables dynamic updates upon any ownership changes in the cloud storage. Whenever an ownership change occurs in the ownership group of outsourced data, the data are re-encrypted with an immediately updated ownership group key, which is securely delivered only to the valid owners. Thus, the proposed scheme enhances data privacy and confidentiality in cloud storage against any users who do not have valid ownership of the data, as well as against an honest-but-curious cloud server. Tag consistency is also guaranteed, while the scheme allows full advantage to be taken of efficient data deduplication over encrypted data. In terms of the communication cost, the proposed scheme is more efficient than the previous schemes, while in terms of the computation cost, taking additional $0.1 - 0.2$ ms compared to the RCE scheme, which is negligible in practice. Therefore, the proposed scheme achieves more secure and fine-grained ownership management in cloud storage for secure and efficient data deduplication.

9. This may happen when the unauthorized users have possessed the data at some time instance and stored the derived key encryption key K until the moment of request; or, they could receive it from the other colluders.

Acknowledgments

This work was supported by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MSIP) (No. 2016R1A2A2A05005402). D. Koo and K. Kang are the corresponding authors of this paper.

References

[1] Dropbox, http://www.dropbox.com/.
[2] Wuala, http://www.wuala.com/.
[3] Mozy, http://www.mozy.com/.
[4] Google Drive, http://drive.google.com.
[5] D. T. Meyer, and W. J. Bolosky, "A study of practical deduplication," Proc. USENIX Conference on File and Storage Technologies 2011, 2011.
[6] M. Dutch, "Understanding data deduplication ratios," SNIA Data Management Forum, 2008.
[7] W. K. Ng, W. Wen, and H. Zhu, "Private data deduplication protocols in cloud storage," Proc. ACM SAC'12, 2012.
[8] M. W. Storer, K. Greenan, D. D. E. Long, and E. L. Miller, "Secure data deduplication," Proc. StorageSS'08, 2008.
[9] N. Baracaldo, E. Androulaki, J. Glider, A. Sorniotti, "Reconciling end-to-end confidentiality and data reduction in cloud storage," Proc. ACM Workshop on Cloud Computing Security, pp. 21–32, 2014.
[10] P. S. S. Council, "PCI SSC data security standards overview," 2013.
[11] D. Harnik, B. Pinkas, and A. Shulman-Peleg, "Side channels in cloud services, the case of deduplication in cloud storage," IEEE Security & Privacy, vol. 8, no. 6, pp. 40–47, 2010.
[12] C. Wang, Z. Qin, J. Peng, and J. Wang, "A novel encryption scheme for data deduplication system," Proc. International Conference on Communications, Circuits and Ssytems (ICCCAS), pp. 265–269, 2010.
[13] Malicious insider attacks to rise, http://news.bbc.co.uk/2/hi/7875904.stm

[14] Data theft linked to ex-employees, http://www.theaustralian.com.au/australian-it/data-theftlinked-to-ex-employees/story-e6frgakx-1226572351953
[15] J. R. Douceur, A. Adya, W. J. Bolosky, D. Simon, and M. Theimer, "Reclaiming space from duplicate files in a serverless distributed file system," Proc. International Conference on Distributed Computing Systems (ICDCS), pp. 617–624, 2002.
[16] P. Anderson, L. Zhang, "Fast and secure laptop backups with encrypted de-duplication," Proc. USENIX LISA, 2010.
[17] Z. Wilcox-O'Hearn, B. Warner, "Tahoe: the least-authority filesystem," Proc. ACM StorageSS, 2008.
[18] A. Rahumed, H. C. H. Chen, Y. Tang, P. P. C. Lee, J. C. S. Lui, "A secure cloud backup system with assured deletion and version control," Proc. International Workshop on Security in Cloud Computing, 2011.
[19] J. Xu, E. Chang, and J. Zhou, "Leakage-resilient client-side deduplication of encrypted data in cloud storage," ePrint, IACR, http://eprint.iacr.org/2011/538.
[20] M. Bellare, S. Keelveedhi, and T. Ristenpart, "Message-locked encryption and secure deduplication," Proc. Eurocrypt 2013, LNCS 7881, pp. 296–312, 2013. Cryptology ePrint Archive, Report 2012/631, 2012.
[21] S. Halevi, D, Harnik, B. Pinkas, and A. Shulman-Peleg, "Proofs of ownership in remote storage systems," Proc. ACM Conference on Computer and Communications Security, pp. 491–500, 2011.
[22] M. Mulazzani, S. Schrittwieser, M. Leithner, and M. Huber, "Dark clouds on the horizon: using cloud storage as attack vector and online slack space," Proc. USENIX Conference on Security, 2011.
[23] A. Juels, and B. S. Kaliski, "PORs: Proofs of retrievability for large files," Proc. ACM Conference on Computer and Communications Security, pp. 584–597, 2007.
[24] G. Ateniese, R. C. Burns, R. Curtmola, J. Herring, L. Kissner, Z. Peterson, and D. Song, "Provable data possession at untrusted stores," Proc. ACM Conference on Computer and Communications Security, pp. 598–609, 2007.
[25] J. Li, X. Chen, M. Li, J. Li, P. Lee, and W. Lou, "Secure deduplication with efficient and reliable convergent key management," IEEE Transactions on Parallel and Distributed Sytems, Vol. 25, No. 6, 2014.
[26] G.R. Blakley, and C. Meadows, "Security of Ramp schemes," Proc. CRYPTO 1985, pp. 242–268, 1985.
[27] J. Li, Y. K. Li, X. Chen, P. 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.
[28] M. Bellare, S. Keelveedhi, T. Ristenpart, "DupLESS: Server-aided encryption for deduplicated storage," Proc. USENIX Security Symposium, 2013.
[29] M. Bellare, S. Keelveedhi, "Interactive message-locked encryption and secure deduplication," Proc. PKC 2015, pp. 516–538, 2015.
[30] Y. Shin and K. Kim, "Equality predicate encryption for secure data deduplication," Proc. Conference on Information Security and Cryptology (CISC-W), pp. 64–70, 2012.
[31] X. Jin, L. Wei, M. Yu, N. Yu and J. Sun, "Anonymous deduplication of encrypted data with proof of ownership in cloud storage," Proc. IEEE Conf. Communications in China (ICCC), pp.224-229, 2013.
[32] D. Naor, M. Naor, and J. Lotspiech, "Revocation and tracing schemes for stateless receivers," Proc. CRYPTO 2001, Lecture Notes in Computer Science, vol. 2139, pp. 41–62, 2001.
[33] K. C. Almeroth, and M. H. Ammar, "Multicast group behavior in the Internet's multicast backbone (MBone)," IEEE Communication Magazine, vol. 35, pp. 124–129, 1997.
[34] Crypto++ Library 5.6.2, http://www.cryptopp.com/.
[35] S. Rafaeli, D. Hutchison, "A Survey of Key Management for Secure Group Communication," ACM Computing Surveys, Vol. 35, No. 3, pp. 309-329, 2003.
[36] L. Mingqiang, C. Qin, P.P.C. Lee, and J. Li, "Convergent Dispersal: Toward Storage-Efficient Security in a Cloud-of-Clouds," Proc. USENIX Conference on Hot Topics in Storage and File Systems, 2014.
[37] L. Mingqiang, C. Qin, P.P.C. Lee, "CDStore: Toward Reliable, Secure, and Cost-Efficient Cloud Storage via Convergent Dispersal," Proc. USENIX Annual Technical Conference, pp. 120, 2015.
[38] J. Li, X. Chen, X. Huang, S. Tang, Y. Xiang, M. Hassan, and A. Alelaiwi, "Secure Distributed Deduplication Systems with Improved Reliability," IEEE Transactions on Computer, Vol. 64, No. 2, pp. 3569–3579, 2015.

Junbeom Hur received the B.S. degree from Korea University, Seoul, South Korea, in 2001, and the M.S. and Ph.D. degrees from KAIST in 2005 and 2009, respectively, all in Computer Science. He was with the University of Illinois at Urbana-Champaign as a postdoctoral researcher from 2009 to 2011. He was with the School of Computer Science and Engineering at the Chung-Ang University, South Korea as an Assistant Professor from 2011 to 2015. He is currently an Assistant Professor with the Department of Computer Science and Engineering at the Korea University, South Korea. His research interests include information security, cloud computing security, mobile security, and applied cryptography.

Dongyoung Koo received the B.S. degree from Yonsei University, Seoul, South Korea, in 2009, and the M.S. and Ph.D. degrees from KAIST in 2012 and 2016, respectively, all in Computer Science. He is currently an Research Professor with the Department of Computer Science and Engineering at the Korea University, South Korea. His research interests include information security, secure cloud computing, and cryptography.

Youngjoo Shin received the B.S. degree in Computer Science and Engineering from Korea University, Seoul, Korea in 2006 and the M.S. and Ph.D. degrees in Computer Science from KAIST, Daejeon, Korea in 2008 and 2014, respectively. He is currently a senior member of engineering staff at the Attached Institute of ETRI, Daejeon, Korea. His research interests include cryptography, network security and cloud computing security.

Kyungtae Kang received the BS degree in computer science and engineering and the MS and PhD degrees in electrical engineering and computer science, from Seoul National University, Korea, in 1999, 2001, and 2007, respectively. From 2008 to 2010, he was a postdoctoral research associate with the University of Illinois at Urbana-Champaign. In 2011, he joined the Department of Computer Science and Engineering, Hanyang University, Korea, where he is currently an assistant professor. His research interests include primarily in systems, such as operating systems, wireless systems, distributed systems, real-time embedded systems, and interdisciplinary area of cyber-physical systems. He is a member of the ACM.