Peer-to-Peer Networking and Applications (2019) 12:850–864
https://doi.org/10.1007/s12083-018-0682-9

Secure deduplication with reliable and revocable key management in fog computing

Hyunsoo Kwon[1] · Changhee Hahn[1] · Kyungtae Kang[2] · Junbeom Hur[1]

Received: 3 July 2018 / Accepted: 5 September 2018 / Published online: 8 October 2018

Abstract
A secure deduplication technique removes duplicate data and stores only single copy to efficiently utilize the storage while guaranteeing the privacy of the data. Thus, it is a necessary technology for resource-limited for devices to save storages. However, most of the existing deduplication schemes based on convergent encryption suffer from 1) a convergent encryption key management problem and 2) a dynamic ownership management problem. In key management, convergent encryption generates a number of encryption keys whose size increases linearly with the number of distinct data. In terms of dynamic ownership management, although the ownership of data in a fog device or cloud storage frequently changes in real-world applications, supporting ownership changes are difficult because the convergent encryption keys are only bound to the data. In order to solve these problems, we present a secure deduplication scheme that features reliable and scalable key management based on pairing-based cryptography and supports dynamic ownership management. The proposed scheme avoids additional costs associated with distributing key components on secure channels and ownership keys on the user side yet guarantees secure key and ownership management.

Keywords Fog computing security · Fault tolerant key management · Dynamic ownership · Secure deduplication

1 Introduction

Fog computing [1, 2] is a promising solution to challenges of the upcoming era, in which numerous IoT devices will be around us and a vast amount of data will come from them. Since fog devices are geographically close to IoT devices, a fog computing paradigm can reduce the latency of data transmission and can promptly process requests received from these devices. Thus, the fog computing is well-suited for various applications, such as healthcare systems that require the immediate processing of protected health information from a variety of wearable devices such as smartwatches, smart bands, and sensors installed to check human health conditions inside a home or hospital [3]. However, as resource-constrained fog devices gather huge volumes of data, including duplicated copies, from IoT devices, there is a need for a method to make efficient use of storage.

The deduplication technique, which eliminates redundant copies and owns only a single copy, is suitable for utilizing a repository efficiently. This approach decreases storage management costs and saves 50–90% of storage [4–6]. In particular, in client-side deduplication, since a fog device uses small pieces of information to inspect the replicated data and the client does not upload any data if it already exists in the fog device, this approach also reduces the use of bandwidth for data upload.

Despite this usefulness, however, users may be concerned about their data privacy because once users upload their data to the fog device, they will not be able to control such data. While this can be clearly solved by outsourcing the encrypted data to the fog device, this strategy may

✉ Kyungtae Kang
ktkang@hanyang.ac.kr

✉ Junbeom Hur
jbhur@korea.ac.kr

Hyunsoo Kwon
hs_kwon@korea.ac.kr

Changhee Hahn
hahn850514@korea.ac.kr

1 Department of Computer Science and Engineering, Korea University, Seoul, 02841, Republic of Korea

2 Department of Computer Science and Engineering, Hanyang University, Ansan 15588, Republic of Korea

conflict with the deduplication technique. Unfortunately, in traditional cryptography, deduplication is not applicable because each user uses different private keys, resulting in different ciphertexts for the same data. To overcome this dilemma, a secure deduplication technique using the convergent encryption (CE) [5] scheme was proposed. In the CE scheme, since the encryption key is derived from the data in a deterministic way, multiple users who have the same data can generate the same encryption key, and the same ciphertext. Therefore, CE can be deployed in a secure deduplication technique. Although many CE-based secure deduplication schemes [11–17] have been proposed, most of the schemes suffer from 1) a convergent encryption key (CEK) management problem and 2) a dynamic ownership management problem.

In terms of the CEK management problem, because the CE generates the same number of CEKs as the number of unique data instances to be encrypted, the user must manage and own a number of CEKs to decrypt ciphertext after obtaining it from a fog device that leverages cloud storage. Anticipating the upcoming high-volume data era, the tremendous increase in the number of encryption keys will be even more problematic for IoT devices. Moreover, if the user does not manage the CEK securely or it is compromised, then his/her ciphertexts may be unrecoverable or leak to malicious users (that is, a reliability problem). In order to support reliable CEK management, a secure deduplication scheme, namely Dekey [17], was proposed. Dekey makes users generate and distribute pieces of CEKs across multiple cloud servers exploiting a secret sharing scheme [18]. When users want to recover the original CEK, they can recover it from a predefined number of pieces instead of storing the entire CEK in the local storage of the user. This can reduce storage costs on the user side and present fault-tolerant key management robust to single-point-of-failure problems. However, Dekey has a strong assumption about the distribution of key shares. If the shares are transmitted over a public channel, anyone who collects a predefined number of shares will be able to access the entire CEK Therefore, in Dekey, users and multiple fog devices (or cloud servers) must set up secure channels using independent cryptographic protocols such as the transport layer security protocol for CEK distribution as well as CEK retrieval. However, according to our experiments, setting up a secure channel increases the computation/communication overhead to a non-negligible level. Since the number of secure channels to be established increases linearly with the number of fog devices, this can incur a scalability problem.

In terms of the dynamic ownership management problem, in a secure deduplication technique, once the users lose their ownership of the original data by deletion or modification, they should no longer be able to decrypt the original ciphertext. Such ownership changes to outsourced data are very common in real-world cloud storage environments [30] and need to be handled carefully to prevent unauthorized access. Hur et al. [11] proposed a secure deduplication scheme that leverages a re-encryption technique of ciphertexts featuring ownership group key management to prevent revoked users from accessing the data. However, their scheme incurs communication overhead linearly as the number of the universe of users increases. In addition, this approach suffers from the CEK management problem.

To resolve these problems, we extend our previous work [31] to present a secure deduplication scheme featuring reliable and scalable CEK management based on pairing-based cryptography, and dynamic ownership management. In our previous scheme [31], for CEK management, we split the CEK into the three key components and allowed only legitimate data owners to obtain the shared secret, which is used as a masking value in the key component distribution, via an independent server. CEK cannot be recovered unless the adversaries who collect key shares on the public channel know this masking value. Furthermore, the three key components are designed to free the user from the burden of managing the key and reduce overall system computation/communication costs. Unlike Dekey, legitimate data owners can retrieve the CEK, and hence the plain data, by eliminating the masking value after collecting key shares from the fog devices without having to establish a secure connection. For dynamic ownership management, in the proposed scheme, we employ re-encryption technique and privilege-based encryption (PE). Cloud storage chooses a random key to re-encrypt the ciphertext encrypted under CE. Then, it encrypts that random key using PE such as attribute-based encryption [33–35], which allows only authorized users to decrypt the re-encrypted data.

The main contributions made by this paper are summarized below:

1. We present a secure deduplication scheme that produces reliable and scalable CEK management. With respect to CEK management, we split the CEK into three key components: a pairing key, shared secret, and partial key. Since this key component management is performed by external entities, it reduces the burden on the user's CEK management. In addition, by leveraging a secret sharing scheme, we improve the reliability of the CEK.
2. Our scheme is resilient against offline brute-force attacks even if the plaintext belongs to a predictable data set. This is accomplished by employing an oblivious pseudo-random function with the key server as in previous schemes, such as DupLess [15].
3. We eliminate the requirement for additional secure channels for distributing the key components by exploiting pairing-based cryptography. Thus, the

privacy of CEKs is protected even when they are transferred through public channels. Therefore, when distributing the CEK, unlike previous scheme, the proposed scheme does not incur extra costs in terms of placing a secure channel.

4. The proposed scheme supports dynamic ownership management. Thus, if users lose their ownership to data, they are prevented from decrypting the original plaintext. According to a security analysis, the proposed scheme guarantees forward/backward secrecy [11] and collusion resistance.
5. We implement the proposed scheme to analyze efficiency. The experimental results indicate that the proposed scheme outperforms the previous schemes in most real-world scenarios. This demonstrates that the proposed scheme is more scalable than the previous ones.

2 Related work

2.1 Convergent encryption

Convergent encryption (CE) allows users to obtain a convergent encryption key (CEK) from the plaintext to be encrypted. If the plaintext is identical, then the same ciphertexts are produced, which is a desirable property for deduplication over ciphertext. Various secure deduplication schemes [14–16] based on CE have been proposed. However, a recent study [15] reported that when the plaintext is predictable, CE's deterministic property makes the ciphertexts vulnerable to brute-force attacks. As a solution, DupLESS [15], produces CEKs in a non-deterministic way by leveraging an oblivious pseudo-random function [19] with the key server so that only valid data owners can get the corresponding CEK without leakage of information of the key. This randomization of CEK prevents brute-force attacks even though the plaintext is chosen from a predictable set. Nevertheless, it also suffers from the CEK management problem. As the number of data instances increases, the overhead needed for secure key distribution impedes the deployment of these schemes in pragmatic settings.

2.2 Reliable deduplication

When employing deduplication techniques, managing only one copy of data implies that its loss can lead to the elimination of all records. To solve this problem, several schemes [13, 17, 21] have been proposed. Among them, Li et al. introduced a secure deduplication scheme called Dekey [17], which utilizes a secret sharing technique to enhance the reliability of the CEK and decrease the overhead for key management. Because anyone can retrieve the CEK by collecting the shares from a public channel, Dekey should send all key shares via additional secure channels. This unavoidable requirement of additional secure channels incurs a non-negligible overhead. Hence, a secure deduplication scheme supporting efficient and scalable key management is required while preserving the privacy of both data and key shares without employing secure channels.

2.3 Deduplication with dynamic ownership management

When a user's ownership of certain data is lost (e.g., by deletion), cloud storage must block the user's access to that data immediately. In CE-based deduplication, however, this is a challenging problem. Because the encryption key is derived from the data, even a revoked user who has lost ownership of the data can still access and decrypt it as long as he/she keeps the encryption key. In order to simultaneously support dynamic ownership management and deduplication, several schemes [11, 32] have been proposed. Since the strategy of handling ownership management in both schemes are the same, we focus on Hur et al.'s scheme, which is the first secure deduplication scheme that prevents revoked users from accessing data through a re-encryption technique using an ownership group key. In their scheme, the user owns the path keys for obtaining the group key. This may incur additional non-negligible storage and communication overhead, especially when the number of users is large. Furthermore, the CEK management problem is inherited in this scheme. Thus, we aim to construct a secure deduplication scheme that presents scalable ownership management while minimizing additional user side overhead.

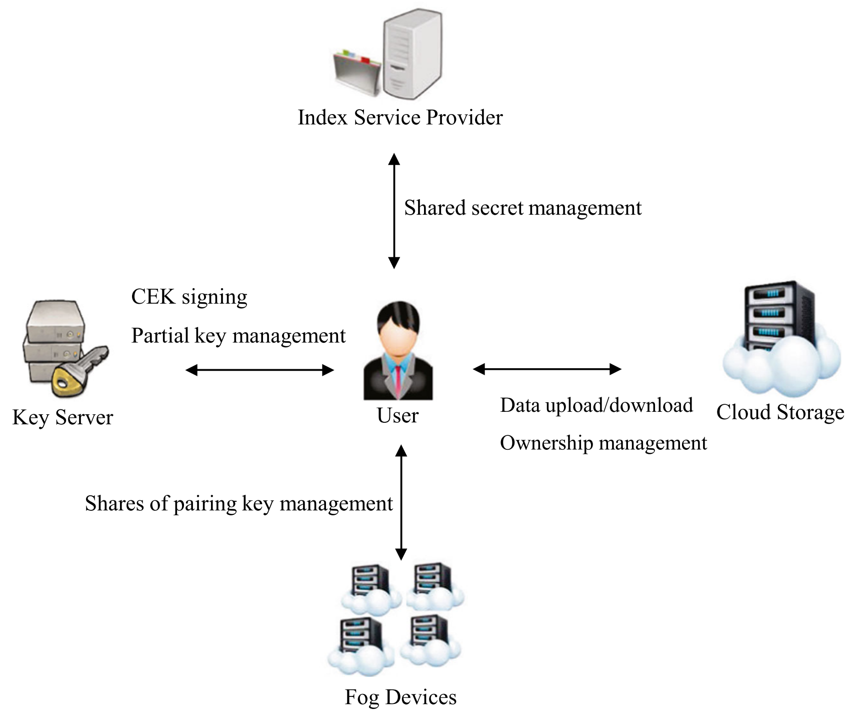
3 System description and security goals

3.1 System model

A system consists of five entities: user, key server (KS), index service provider (IS), cloud storage (CS), and fog devices (FDs). Figure 1 illustrates the system architecture of the fog computing system.

A **user** desires to upload his/her data into CS in order to save on local storage. Before outsourcing the data, he/she encrypts it using CE for deduplication of ciphertext. The user cooperates with the other entities to manage the CEK reliably, efficiently, and securely. In addition, the user owns the privilege key corresponding to his/her ID for user dynamic ownership management. When a user eliminates

Fig. 1 System model



the ciphertext in CS, his/her privilege key becomes invalid for that data, so the revoked user can no longer decrypt the ciphertext even with the CEK.

The **Key Server (KS)** takes two different tasks from the system. First, it signs the CEK to improve the privacy of the outsourced data against an offline brute-force attack. Second, it maintains the partial keys obtained from users to diminish overhead for managing the CEK.

The **Index Service Provider (IS)** creates the shared secret and transmits it to users so that those who upload the same data receive the same shared secret. In addition, the IS maintains the index idx of each CEK to allow both the KS and the IS to hold one key component corresponding to the data, and checks whether a user is authorized to obtain the idx.

Cloud Storage (CS) presents a data outsourcing service to users and manages the data on their behalf. CS employs a deduplication technique to avoid duplicated data storage. In terms of ownership management, CS generates a privilege key for each registered user and encrypts the random key, which is used to re-encrypt the ciphertext using privilege-based encryption. In this paper, we state that CS manages the outsourced data and ownership.

Fog Devices (FDs), in terms of CEK management, maintain secret shares of pairing key and distribute them to users for reliable and efficient key management. Since FDs are geographically close to the user, the network latency between them can be reduced compared to the communication between the user and CS. Therefore, they can respond more rapidly to a request of the pairing key shares of the users.

3.2 Threat model

We assume that there are two kinds of adversary 1) an inside adversary and 2) an outside adversary. An inside adversary is a legal system entity, and is thus allowed to access the data it stores and manages. In our system model, the CS, FD, and KS are regarded as potential inside adversaries. Inside adversaries who are considered curious-but-honest perform their tasks accurately but desire to know the secret data of the user. In addition, CS, FD, and the KS can conduct a collusion attack to reveal the user's CEK.

An outside adversary wants to illegally acquire useful information about users' data and the random keys chosen by the CS. They can cooperate with a legal user whose access rights are valid to acquire private data. Outside adversaries who are considered malicious users try to obtain original data that he/she does not own without proper privilege to satisfy the access policy of the ciphertext.

The adversaries can undertake a collusion attack with other users who do not have access rights in order to obtain the key used for re-encryption by combining their invalid privilege keys to satisfy the access policy.

We assume that the IS is fully trusted and complies with the specification of the proposed scheme. In other words, the IS does not collude with any insider or outside adversaries. As the IS must identify authorized users who have the data from adversaries who do not and thus selectively provide the shared secret, it must remain a trusted entity. The presence of a trusted entity has been approved in various fields [23–25] to accomplish varied security goals and is also required by the secure deduplication scheme [14].

3.3 Security goal

We aim to achieve four security goals: data security, convergent encryption key security, forward/backward secrecy [11], and collusion resistance.

Data confidentiality and integrity The proposed scheme has to guarantee the privacy of outsourced data against adversaries who do not have the same data in CS. Data confidentiality must be resilient even though the adversaries try to conduct a dictionary attack on plaintext that is selectable from a predictable data set. In order to guarantee the integrity of the data (or tag consistency [10]), our scheme must distinguish and block duplicate faking attacks [10]. These attacks arise when adversary outsources maliciously produced ciphertext c_1 instead of honestly encrypted data c_2 such that the tag of c_1 is equal to the tag of c_2. Thus, through deduplication the user who outsourced c_2 might receive the wrong ciphertext c_1.

Convergent encryption key security We divide the CEK into three key components: the pairing key, the shared secret, and the partial key. Each key component is sent to three different types of entities. The pairing key is divided into n shares, which are transmitted to n FDs. The shared secret and partial key are transferred to the IS and KS, respectively. The proposed scheme should protect the privacy of these three key components despite their being sent on public channels and although the adversaries collude with semi-trusted entities such as KS, CS, or FD.

Forward and backward secrecy [11] For secure dynamic ownership management, the proposed scheme should ensure forward/backward secrecy. In the context of ownership management for deduplication, forward secrecy means preventing revoked users who have deleted or modified the data from decrypting the ciphertext. Conversely, backward secrecy means preventing users who outsource the same data from decrypting the corresponding ciphertext before they have proven their ownership.

Collusion resistance The proposed scheme should forbid unauthorized users who do not have legitimate data ownership from decrypting and obtaining the original data even if they conduct a collusion attack using their privilege key.

4 Proposed scheme

4.1 Preliminaries

We commonly denote variables by lowercase letters (e.g., a, b, c), algorithms by uppercase letters (e.g., A, B, C), and sets by blackboard bold letters (e.g., $\mathbb{G}, \mathbb{Z}$). Additionally, we denote by $a \leftarrow A$ the assignment of a as the deterministic result of algorithm A, and denote by $a \in_R \mathbb{A}$ the assignment of a as a random selection from the set $\mathbb{A}$. The one-time symmetric encryption SE [37] consists of a deterministic encryption algorithm Enc_{SE} and a decryption algorithm Dec_{SE}. With the random key $K \in_R \{0, 1\}^k$ and data M, the encryption and decryption algorithms output the ciphertext $C = Enc_{SE}(M, K)$ and plaintext $P = Dec_{SE}(C, K)$, respectively.

4.1.1 Bilinear maps

Let $\mathbb{G}$ and $\mathbb{G}_T$ be two multiplicative cyclic groups of prime order p, and g be a generator of $\mathbb{G}$. Let a bilinear map $\hat{e}$ be a map $\hat{e}$: $\mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$, with the following properties:

– Bilinearity: For all $u, v \in \mathbb{G}$ and $a, b \in \mathbb{Z}_p$, $\hat{e}(u^a, v^b) = \hat{e}(u, v)^{ab}$
– Computability: There exists an efficient algorithm to compute map $\hat{e}$
– Non-degeneracy: $\hat{e}(g, g) \neq 1$.

4.1.2 Convergent encryption

The CE scheme [10] consists of the following four algorithms: Gen_{CE}, Enc_{CE}, Dec_{CE}, Gen_{Tag}.

– $ck \leftarrow Gen_{CE}(D)$: Taking data D, it outputs CEK $ck \leftarrow H(D)$, where $H(\cdot)$ is the cryptographic hash function.
– $C \leftarrow Enc_{CE}(D, ck)$: Taking D and ck, it encrypts D using a symmetric and deterministic encryption scheme with the ck generated from Gen_{Key} and outputs the ciphertext C.
– $D \leftarrow Dec_{CE}(C, ck)$: Taking C and ck, it decrypts C inversely and outputs the plaintext D.
– $T \leftarrow Gen_{Tag}(C)$: Taking C, it outputs tag $T \leftarrow H(C)$.

4.1.3 Oblivious-pseudo random function

To sign the hash value we employ an oblivious-pseudo random function (O-PRF) protocol based on the RSA blind signature [15]. The O-PRF consists of the following four algorithms: Gen_{RSA}, $Sign_U$, $Sign_S$, Ver_{Sign}.

– $(pk_s, sk_s) \leftarrow Gen_{RSA}(e)$: Taking RSA exponent e, it outputs the KS's key pair $(pk_s, sk_s) =$

$((N, e), (N, d))$, where $ed \equiv 1 \bmod \phi(N)$, N is the product of two distinct prime numbers.

- $\sigma \leftarrow Sign_U(D, pk_s) \overset{\$}{\rightarrow} Sign_S(sk_s)$: The two interactive algorithms take (D, pk_s) and sk_s as inputs, respectively. $Sign_U$ run by the data owner derives the hash value h from the data D and randomly chooses the value $r \in_R \mathbb{Z}_{\mathbb{N}}{}^*$. Then, the resulting blinded hash value $x = h \cdot r^e \bmod N$ is fed into $Sign_S$. Upon receiving x, $Sign_S$ run by the KS computes $y = x^d \bmod N$ and returns y to $Sign_U$. The result of the two interactive algorithms is a signed hash value $\sigma = y \cdot r^{-1} \bmod N = (h \cdot r^e)^d \cdot r^{-1} \bmod N = h^d \bmod N$.
- $(\sigma/\perp) \leftarrow Ver_{Sign}(\sigma, h)$: Taking the signed hash value σ and h, it outputs σ if $h = \sigma^e \bmod N$, $\perp$ otherwise.

4.1.4 Proofs of ownership

The proofs of ownership (PoW) enables the user to efficiently verify the ownership of the outsourced data without delivering all the data.

In PoW, the prover who claims to have access to the outsourced data constructs a Merkle tree. The prover divides the entire set of data into multiple blocks and uses a collision-resistant hash function to produce the hash values of the grouped blocks in pairs. It then groups the hash values iteratively into pairs and hashes each pair from the leaf nodes into the root node. After repeating this process, the prover can earn a single hash value for all of the data, which matches to the root node and all the blocks derived from the data corresponding to the leaves.

The verifier keeps the root and the number of leaves in the Merkle tree, while the prover maintains the entire Merkle tree. To check that the prover indeed owns the data, the verifier randomly selects r leaf indexes as a challenge message and transmits them to the prover. Then, the prover collects the sibling-path which consists of all hash values needed to generate the root hash value from the particular leaves matching to each leaf index in the challenge message, and sends it to the verifier. After obtaining the sibling paths for the challenge message, if the verifier can construct the root hash value from them, it ensure that the prover really has the data. Otherwise, the request from the prover is denied.

4.1.5 Ramp secret sharing scheme

We employ the (n, k, r)-secret sharing scheme, which produces n shares on the secret value, as an input. This scheme satisfies the following two conditions: (1) The secret value is restorable when the number of collected shares is greater than or equal to k; otherwise, it is not restorable. (2) No one can deduce any information about secret values from the shares less than r, where $n > k > r \geq 0$. The secret sharing scheme consists of two algorithms: Share and Recover.

Share$(\cdot)$ splits an input value into $(k - r)$ shares and randomly selects r shares. It then encodes the k shares into n shares, and finally outputs n shares. However, since each share is randomly selected, which obstructs share deduplication, it is not applicable to our scheme. Therefore, we create the r shares using the pseudo-random approach as in Dekey [17].

Recover$(\cdot)$ outputs the original secret value on input k of n shares.

4.1.6 Privilege-based encryption

In conventional symmetric and asymmetric encryption, data can be encrypted to only a designated user. In order to overcome this limitation, privilege-based encryption (PE) was proposed. PE encrypts data for an arbitrary set of users. There are various types of PE, such as identity-based encryption (IBE) [36], attribute-based encryption (ABE) [35], and ciphertext/key-policy attribute-based encryption (CP/KP-ABE) [33, 34]. Since the ciphertext is generated on the basis of a set of privileges, PE can enforce access policies on the ciphertext without prior knowledge of the recipients. In the proposed scheme, we employ the PE scheme, especially CP-ABE [33], which consists of the following four algorithms: $Setup_{PE}$, Gen_{PE}, Enc_{PE}, Dec_{PE}.

- $P_C \leftarrow Setup_{PE}(\lambda)$: Taking security parameter λ, it generates the master key P_C owned by CS.
- $P_U \leftarrow Gen_{PE}(Set_U)$: Taking Set_U as the set of users who are registered to CS, it generates privilege keys P_U for each user's ID. CS then distributes them to the registered users.
- $C \leftarrow Enc_{PE}(D, A)$: Taking D and the access policy A constructed by CS, it encrypts D using a privilege-based encryption and outputs the ciphertext C.
- $D \leftarrow Dec_{PE}(C, P_U)$: Taking C and P_U, it decrypts C and outputs the plaintext D as long as P_U satisfies A.

4.2 Scheme construction

4.2.1 System setup

A user generates his/her private and public key pair (sk, pk). The user randomly chooses $x \in_R \mathbb{Z}_p{}^*$ as a secret key sk and obtains public key $pk = g^{x^{-1}}$. The KS owns its secret key $sk_s = (d, N)$ and public key $pk_s(e, N)$ by running Gen_{RSA}. To leverage PE, the CS runs $Setup_{PE}(\lambda)$ and $Gen_{PE}(Set_U)$, where λ is the security parameter in the system, and Set_U is the set of the users in the system.

4.2.2 Data upload process

The user computes the CEK by following the O-PRF protocol with the KS. Then, he/she obtains the ciphertext using CE with the CEK and computes its tag in order to check whether a duplicate copy exists in CS.

After uploading the ciphertext, the user starts the CEK distribution process, and the CS re-encrypts the data using privilege-based encryption with a randomly selected key for ownership management. Figure 2a illustrates the data upload process, which is described as follows:

Generate a signed hash value h^d: To acquire the CEK, a user follows the O-PRF protocol with the KS. $Sign_U$, run by the user, interacts with the $Sign_S$ run by the KS for given input data D and the key pair of KS (sk_s, pk_s), and outputs σ. If σ is accepted by Ver_{Sign} for given inputs σ and hash value h, σ can considered as a signed hash value h^d.

Outsource the ciphertext and its tag: After generating h^d, the user can derive a CEK $ck \leftarrow G(h^d)$, where G: $\mathbb{Z}_N^* \rightarrow \{0,1\}^*$. The user obtains the ciphertext $C \leftarrow Enc_{CE}(D, ck)$ and computes tag $T \leftarrow H(C)$. Then, he/she transmits T to CS to check whether it is duplicated.

Identify the duplicate copy: On receiving T, CS tests for the existence of a duplicate record and returns the resulting of checking to the user. If there is a duplicate copy, the user begins the CEK distribution process. Otherwise, the user transmits C and T to CS. After passing the test for checking whether T was precisely generated from C, the user embarks on the CEK distribution phase. Otherwise, CS denies the request for uploading the data.

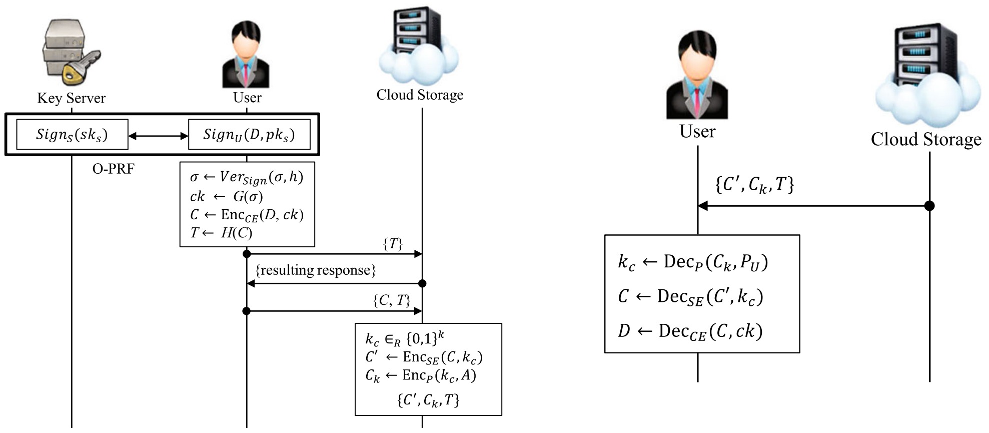
Re-encrypt the outsourced data: To support dynamic ownership management for an outsourced ciphertext C, when the user uploads the ciphertext to CS, CS randomly chooses a key $k_c \in_R \{0,1\}^k$ and obtains ciphertext $C' \leftarrow Enc_{SE}(C, k_c)$. Then, CS defines the access policy A with the set of authorized users and gets encrypted key $C_k \leftarrow Enc_P(k_c, A)$.

4.2.3 CEK distribution

In the CEK distribution phase, the user generates the pairing key and transmits it to the IS. Then, the IS computes a blinded shared secret and returns it to the user. After taking it, the user deduces the shared secret. (All owners of the same data will obtain the same secret at this point.) The user computes the partial key and sends it to the KS. Finally, the user splits the pairing key into n shares exploiting the secret sharing scheme, and transmits n shares to n distributed FDs. Figure 3a illustrates the CEK distribution process. The CEK distribution procedure is described as follows:

Generate the pairing key pak: The user computes the pairing key $pak = \hat{e}(g,g)^{sh}$, where $sh \leftarrow F(h^d)$ and $F : \mathbb{Z}_N \rightarrow \mathbb{Z}_p$, and transmits it to the IS.

Identify the authorized user: On receiving pak, the IS computes an index $idx \leftarrow H(pak)$ and retrieves idx from the index database. If idx exists, it means that the user has uploaded duplicate data to CS. In this case, he/she is not obliged to send the CEK because someone who uploaded the same data has already created and sent the CEK to FD. Then, the IS uses the proofs-of-ownership (PoW) protocol [22] to check whether he/she owns the data corresponding to idx or not. After that, only an authorized user is enrolled in the database and finishes the CEK distribution successfully. Otherwise, the requests from unauthorized users are denied. If idx is not detected, the user (considered an initial uploader) transmits the root hash value of the Merkle tree of the

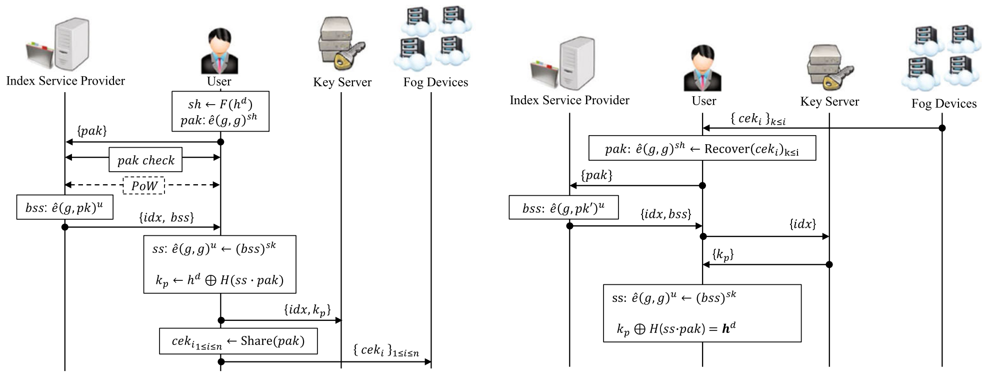


Key Server
User
Cloud Storage
$Sign_S(sk_s)$
$Sign_U(D, pk_s)$
O-PRF
$\sigma \leftarrow Ver_{Sign}(\sigma, h)$
$ck \leftarrow G(\sigma)$
$C \leftarrow Enc_{CE}(D, ck)$
$T \leftarrow H(C)$
$\{T\}$
{resulting response}
$\{C, T\}$
$k_c \in_R \{0,1\}^k$
$C' \leftarrow Enc_{SE}(C, k_c)$
$C_k \leftarrow Enc_P(k_c, A)$
$\{C', C_k, T\}$
User
Cloud Storage
$\{C', C_k, T\}$
$k_c \leftarrow Dec_P(C_k, P_U)$
$C \leftarrow Dec_{SE}(C', k_c)$
$D \leftarrow Dec_{CE}(C, ck)$

(a) Data upload process (b) Data download process

Fig. 2 Data upload/download process

Fig. 3 CEK management


Index Service Provider
User
Key Server
Fog Devices
$sh \leftarrow F(h^d)$
pak: $\hat{e}(g,g)^{sh}$
{pak}
pak check
PoW
bss: $\hat{e}(g,pk)^u$
{idx, bss}
ss: $\hat{e}(g,g)^u \leftarrow (bss)^{sk}$
$k_p \leftarrow h^d \oplus H(ss \cdot pak)$
{idx, k_p}
$cek_{i_{1\le i\le n}} \leftarrow$ Share(pak)
{ cek_i }$_{1\le i\le n}$
{ cek_i }$_{k\le i}$
pak: $\hat{e}(g,g)^{sh} \leftarrow$ Recover$(cek_i)_{k\le i}$
{pak}
bss: $\hat{e}(g,pk')^u$
{idx, bss}
{idx}
{k_p}
ss: $\hat{e}(g,g)^u \leftarrow (bss)^{sk}$
$k_p \oplus H(ss{\cdot}pak) = \boldsymbol{h}^d$

(a) CEK distribution (b) CEK retrieval

ciphertext to the IS. The IS then selects the random value $u \in_R \mathbb{Z}_p$ and generates the blinded shared secret $bss = \hat{e}(g, pk)^u$ using the user's public key pk so that only authorized users can obtain the shared secret from bss. After that, it sends bss and idx to the authorized user, who uploads the non-redundant data and enrolls the user in the database.

Compute the partial key k_p: On receiving idx and bss, the user obtains the shared secret $ss = \left(\hat{e}(g, pk)^u\right)^{sk} = \hat{e}(g, g)^u$ and generates the partial key $k_p \leftarrow h^d \oplus H(ss \cdot pak)$, where H is the cryptographic hash function. The partial key is needed to obtain the CEK in the CEK retrieval phase. After transmitting k_p that connotes h^d to the KS with idx, to use the storage efficiently, he/she then removes the partial key from his/her local storage. The KS then maintains idx and k_p in the database.

Distribute n **shares**: In order to enhance the reliability of the CEK, by leveraging the secret sharing scheme, the user has divided pak into n shares. For a given input pak, the user runs the Share(·) function and acquires $cek_{i\,(1\le i\le n)} \leftarrow$Share($pak$). After that, the user distributes the shares cek_i to n FDs on a public channel and terminates the CEK distribution process successfully.

4.2.4 CEK retrieval

In order to obtain the original data from the ciphertext, the user has to retrieve the CEK from some of the secret shares cek_i, which are maintained in n of distributed FDs. According to a property of the secret sharing scheme, after the user collects k of n shares from FD, he/she can restore the CEK from it. In other words, even if (n-k) of the FDs lose their shares, the CEK can still be retrieved from the collected k shares. Figure 3b illustrates the CEK retrieval process. The CEK retrieval procedure is described as follows:

Collect k **of** n cek_i: After collecting k of n cek_i from FD, the user uses the Recover(·) function to restore $pak = \hat{e}(g, g)^{sh} \leftarrow \text{Recover}(cek_i)_{(k\le i)}$.

Obtain the signed hash value h^d: The following procedure is similar to the process of CEK distribution. To acquire the index idx and blinded shared secret $bss = \hat{e}(g, pk)^u$, the user transmits pak to the IS. After that, only authorized users registered in the database can obtain the bss and idx, and generate the shared secret $ss = \hat{e}(g, g)^u$. The user then sends idx to the KS and receives the partial key k_p corresponding to idx. Subsequently, he/she can obtain the signed hash value h^d by the calculation below:

$$\begin{aligned} & k_p \oplus H(\hat{e}(g, g)^{sh} \cdot \hat{e}(g, g)^u) \\ & = h^d \oplus H(\hat{e}(g, g)^{sh} \cdot \hat{e}(g, g)^u) \oplus H(\hat{e}(g, g)^{sh} \cdot \hat{e}(g, g)^u) \\ & = h^d. \end{aligned}$$

Now the user computes $ck \leftarrow G(h^d)$ and is ready to decrypt the ciphertext downloaded from CS.

4.2.5 Data download process

In the data download phase, the user downloads the ciphertext pair C', C_k and tag T from the CS. The user who has a valid privilege key P_U, which means his/her P_U satisfies the embedded access policy A, can obtain $k_c \leftarrow Dec_{PE}(C_k, P_U)$. After that, the user computes $C \leftarrow Dec_{SE}(C', k_c)$, which is the original ciphertext encrypted by CE. To decrypt C, the user has to restore the CEK in the CEK retrieval process, as explained in the previous subsection. After computing the CEK ck, he/she can obtain the original plaintext $D \leftarrow Dec_{CE}(C, ck)$ from the ciphertext C. Figure 2b illustrates the data downloading process.

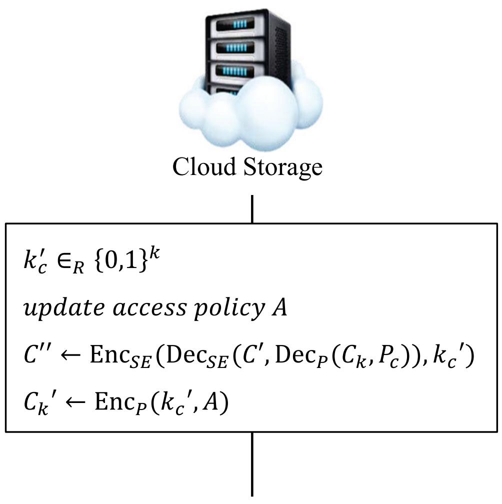

Cloud Storage
$k_c' \in_R \{0,1\}^k$
update access policy A
$C'' \leftarrow \mathrm{Enc}_{SE}(\mathrm{Dec}_{SE}(C', \mathrm{Dec}_P(C_k, P_c)), k_c')$
$C_k' \leftarrow \mathrm{Enc}_P(k_c', A)$

Fig. 4 Encrypted key update

4.2.6 Encrypted key update

To manage the dynamic ownership securely, CS performs an encrypted key update when a new user uploads redundant data or an existing owner deletes his data. If a joining or revocation event occurs, CS first selects the new random key $k_c' \in_R \{0, 1\}^k$ and updates the access policy A with the new set of valid users. After that, CS refreshes the ciphertext pair $C'' \leftarrow Enc_{SE}(Dec_{SE}(C', Dec_{PE}(C_k, P_c)), k_c')$ and encrypted key $C_k' \leftarrow Enc_{PE}(k_c', A)$ using the privilege-based encryption scheme. Since the ciphertext pair is re-encrypted with newly generated parameters such as $k_c'$and A, only a user with a qualified privilege key is allowed to obtain and decrypt the ciphertext in CS. Figure 4 illustrates the encrypted key update process.

5 Analysis

5.1 Comparison of secure deduplication schemes

Table 1 compares CE-based secure deduplication schemes in terms of data deduplication with regard to ciphertext, reliable key management, and ownership management . All of the above schemes allow CS to perform deduplication on ciphertexts by leveraging CE. Dekey [17] and the proposed scheme allow users to distribute CEK shares on multiple FDs and recover the original CEK using a secret sharing scheme, so reliable key management is guaranteed. Regarding ownership management, whenever any ownership for some data changes, Hur et al.'s scheme and the proposed scheme can immediately update the ownership list and re-encrypt that data. Thus, only valid users can decrypt the ciphertext.

Table 1 Comparison results of secure deduplication schemes

Scheme	Deduplication on ciphertext	Reliable key management	Ownership management
Dekey [17]	Yes	Yes	No
Hur et al. [11]	Yes	No	Yes
Proposed	Yes	Yes	Yes

5.2 Security analysis

We analyze the security of the proposed scheme in terms of the security requirements defined in Section 3.3.

5.2.1 Data confidentiality and integrity

In terms of data confidentiality, there are two types of adversaries (outside adversaries and inside adversaries) who desire to disclose the outsourced data that they do not have in CS. Outside adversaries, despite being able to acquire the signed blinded hash value $y = h^d \cdot r \bmod N$ through public channels, cannot obtain the signed hash value h^d without knowledge of r, which is a random value selected by an authorized user.

Furthermore, since the hash value of the plaintext, used as an encryption key, looks random to the outside attacker by running O-PRF with the KS, the proposed scheme is secure against an offline brute-force attack even if the data is predictable. Although outside adversaries guess some hash values for predictable data, KS prevents them from obtaining the signed hash value by masking it.

We can regard inside adversaries as being stronger adversaries than outside adversaries since they can additionally exploit the information they control, such as the KS's private key d. Then, the proposed scheme is no longer secure against offline brute-force attacks against data selected from the predictable set because they can generate the signed hash values of the predictable data. Nevertheless, inheriting DupLESS [15] can guarantee the security of the CE, so the proposed scheme provides data privacy against offline brute-force attacks on the data selected from unpredictable set [10].

In terms of data integrity (that is, tag consistency), a user derives a tag from the ciphertext rather than plaintext before uploading it. Since the tag is generated from the encrypted data, CS can determine whether it is derived from the corresponding ciphertext or not. If it does not match, CS denies the uploading request. Moreover, on receiving the ciphertext pair from CS, it also allows the user to check the association between the ciphertext and its tag. Because both the user and CS can identify the correct tag in the fake tag, our scheme provides data integrity for the outsourced data by employing tag consistency [10].

5.2.2 Convergent encryption key security

In the proposed scheme, CEK is divided into three key components that are sent on public channels. In this section, we determine whether our scheme provides privacy for the three key components against attackers even when they collude with the CS, FD, or KS.

Pairing key For a given input h^d, the user computes the pairing key $pak = \hat{e}(g, g)^{sh}$, where $sh \leftarrow F(h^d)$. Then, he/she splits pak into n shares by employing a secret sharing scheme and sends them to n distributed FDs. Although after the adversaries have gathered k of n shares, the information restored from the shares is only the pairing element $\hat{e}(g, g)^{sh}$. Under the discrete logarithm assumption, the adversary cannot deduce sh from the $\hat{e}(g, g)^{sh}$. Thus, it prevents the exposure of secret information and guarantees the security of the pairing key even if the shares are sent via public channels.

Shared secret The IS generates the shared secret $\hat{e}(g, g)^u$ using pairing exponentiation and distributes it to allow only authorized users who have the same data in CS to obtain a secret value from the blinded shared secret. When generating the shared secret, by using the requested user's pk, the IS ensures that the shared secret is not exposed by masking it as follows: $\hat{e}(g, pk)^u = \hat{e}(g, g)^{\frac{u}{x}}$, where $(sk, pk) = (x, g^{x^{-1}})$ is the user's key pair. At this step, the IS confirms that the user who asks the shared secret has the data corresponding to the pairing key using the PoW [22] protocol. By leveraging the PoW protocol, the IS can prevent the shared secret from being leaked to a malicious user who has no data in CS. Since the hardness of obtaining the shared secret $\hat{e}(g, g)^u$ from the blinded shared secret is equivalent to solving the discrete logarithm assumption even if the blinded shared secret is sent on public channels, the proposed scheme guarantees the privacy of the shared secret unless the private key of the user is compromised.

Partial key The partial key is generated from the CEK, the pairing key, and the shared secret by using a group operation on $\mathbb{G}_T$. If the adversaries who do not have the data cannot obtain the shared secret from the blinded shared secret, then they cannot obtain the signed hash value h^d. The partial key's security is guaranteed as long as the shared secret is not revealed to the adversary.

Thus, our scheme provides privacy for the three key components of the CEK even when they are sent via public channels.

5.2.3 Forward and backward secrecy

In terms of forward secrecy, a user who loses his/her access rights to the encrypted data in CS should be prevented from obtaining the corresponding plaintext. When the user deletes or modifies the outsourced data, CS updates the access policy A with a set of the valid user's privilege keys and re-encrypts the ciphertext from C' to C'' with a randomly selected key k'_c. Then, k'_c is securely distributed to the set of valid owners using PE under the updated A. Thus, the revoked user cannot decrypt such re-encrypted ciphertext because his/her privilege key cannot satisfy A that has been updated. Even if the users recover CEK through the CEK retrieval process and hide it before losing ownership, they are unable to acquire the plaintext because they cannot decrypt the ciphertext that has been encrypted by CS. Thus, the proposed scheme ensures the forward secrecy of the data outsourced to CS.

In terms of backward secrecy, a user who desires to upload redundant data that exists in CS should be prevented from obtaining the plaintext of that data before taking ownership. When the user outsources the duplicated data, CS performs ownership management for that data as above and maintains the updated ciphertext pair $C'' \leftarrow Enc_{SE}(Dec_{SE}(C', Dec_{PE}(C_k, P_c)), k'_c)$ and $C'_k \leftarrow Enc_{PE}(k'_c, A)$. Then, although the user may obtain the previous ciphertext C', it cannot be decrypted since his/her privilege key cannot meet the policy A embedded in C'. Only users who have undergone ownership management can access and decrypt that data. Thus, the proposed scheme ensures the backward secrecy of the outsourced data in CS.

5.2.4 Collusion resistance

To ensure the collusion resistance of the proposed scheme, unauthorized users should be prevented from decrypting the data with their invalid privilege keys even if they collude. In the CP-ABE encryption scheme employed for PE, the privilege keys are constructed with a personalized random value. Even if a malicious user combines his/her privilege key with that of others to satisfy the access policy embedded in the ciphertext, he or she cannot decrypt and obtain the plaintext because each random number encapsulated in the privilege keys is different [33]. Thus, the proposed scheme is secure against collusion attacks.

5.3 Performance analysis

We introduce the results of a performance analysis of our scheme as compared to Dekey in terms of reliable key management and Hur et al.'s scheme [11] in terms of

Table 2 Computation costs

Scheme	CEK management		Ownership management	
	CEK distribution	CEK retrieval	Data re-encryption	Key re-encryption
Dekey [17]	N$\cdot(Sha + R_E + R_D)$	N$\cdot(R_E + R_D)$	–	–
Hur et al. [11]	–	–	$E_S + D_S$	G$\cdot E_S$
Proposed	$P + 2P_E + M_G + X_G$	$P_E + M_G + X_G$	$E_S + D_S$	$E_P + D_P$

ownership management. Regarding CEK management, we assume that cloud storage servers in Dekey play the role of fog devices for a fair comparison.

5.3.1 Computation costs

Table 2 lists the comparison results of the computation costs of the three schemes. In Table 2, the notations are defined as follows: P and P_E are the bilinear map function and the pairing exponentiation, respectively, M_G and X_G are multiplication and XOR operations on group $\mathbb{G}_T$, Sha is the SHA-256 hash function, R_E and R_D are RSA encryption/decryption, E_P and D_P are privilege-based encryption/decryption, and E_S and D_S are symmetric encryption/decryption, respectively.

In terms of CEK management, the computation costs for building secure channels grow with the number of FDs because Dekey must set additional secure channels with distributed FDs to transmit the key shares to guarantee their privacy. By contrast, our scheme provides a consistent computation cost regardless of the number of FDs because independent secure channels for distributing and restoring key shares are not needed.

In terms of ownership management, Hur et al.'s scheme [11] and the proposed scheme have a single symmetric encryption/decryption cost for re-encrypting the ciphertext. In addition, Hur et al.'s scheme has to encrypt the key G times using symmetric encryption, while the proposed scheme performs a single PE encryption/decryption, which is more computationally expensive than symmetric encryption.

5.3.2 Communication costs

Table 3 compares the communication costs of the three schemes. First, with respect to CEK management, the elements that determine the cost of Dekey are consistently affected by the number of FDs.

Based on Table 4, the communication costs are measured in bytes and we can acquire the communication costs in Dekey and the proposed scheme as follows:

$$\text{Proposed scheme}: (N\times 512)+(\log_2(data/32)\times 32)+96$$
$$\text{Dekey}: N\times\{(\log_2(data/32)\times 32)+192\}.$$

Figure 5a and b illustrate the communication costs of CEK management using the above equations when the number of FDs holds at 5 or 15 with different data sizes (that is, 2^6KB, 2^7KB, 2^8KB, 2^9KB or 2^{10}KB). According to Fig. 5a and b, our scheme has better communication performance than that of Dekey if the size of the data is greater than 312KB when there are 5 FDs and 64KB when there are 15 FDs. Thus, the size of the data for which our scheme can be more scalable and efficient than Dekey decreases as the number of FDs increases.

Next, we compare the communication costs related to personal keys that are necessary for ownership management on the client side with different numbers of users. Figure 5c illustrates the communication cost of ownership key distribution as the number of the universe of users in the system varies from 5000 to 25000 when the number of data owners is 10. As shown in Fig. 5c, the communication

Table 3 Communication costs

Scheme	CEK distribution	CEK retrieval	Ownership key distribution
Dekey [17]	N $\cdot S_{Enc(s)}$, N $\cdot S_{root}$, N $\cdot S_{PoW}$, N $\cdot S_{tag}$	N $\cdot S_{Enc(s)}$	–
Hur et al. [11]	–	–	(U−M)$\cdot \log_2\left(\frac{U}{U-M}\right)\cdot S_{kek}$
Proposed	S_{PoW}, N $\cdot S_{share}$, S_{idx}, S_{pair}, S_{root}, S_{part}	S_{pair}, S_{idx}, S_{part}, N $\cdot S_{Enc(s)}$	S_{ek}+M$\cdot S_{attr}$

Table 4 Notations and their sizes

Notation	Description	Size (Byte)
N	Number of FDs that maintain the shares	–
U	Number of users in the system	–
M	Number of owners in an ownership list for a file	–
G	Number of KEKs selected for ownership management	–
S_{sk}, S_{pk}	Size of user's private key and public key	16
S_{pair}	Size of pairing element	16
S_{part}	Size of partial	16
S_u	Size of common secret value	16
S_{idx}	Size of index	32
S_{tag}	Size of tag	32
$S_{Enc(s)}$	Size of ciphertext of key share	128
S_{share}	Size of key share	512
S_{PoW}	Size of overall hash block used in PoW	$32\times \log_2\left(\frac{data}{32}\right)$
S_{root}	Size of root value of Merkle tree	32
S_{kek}	Size of KEK	16
S_{ek}	Size of encrypted key	324
S_{attr}	Size to increase when adding attributes	274

overhead of Hur et al.'s scheme increases linearly because it is affected by the number of users registered in the system. Even if the number of data owners is the same, the overhead depends on the total number of users. By contrast, our scheme has fixed communication overhead regardless of the number of universe of users in the system.

Therefore, with respect to CEK management and ownership management communication costs, our scheme

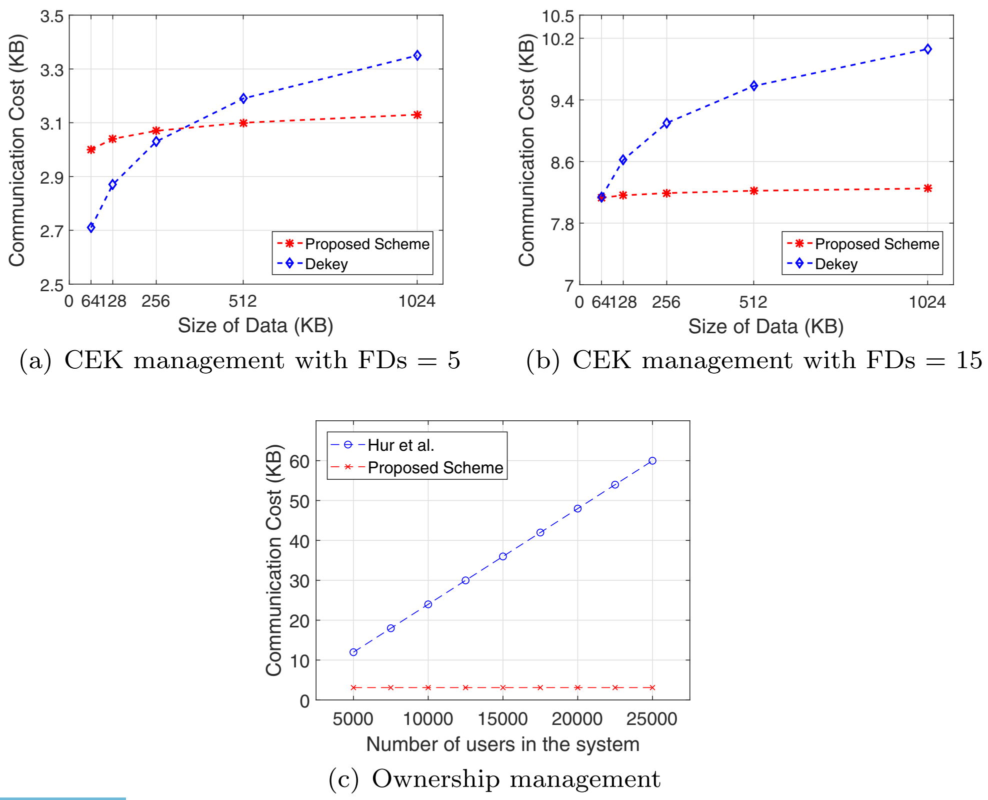
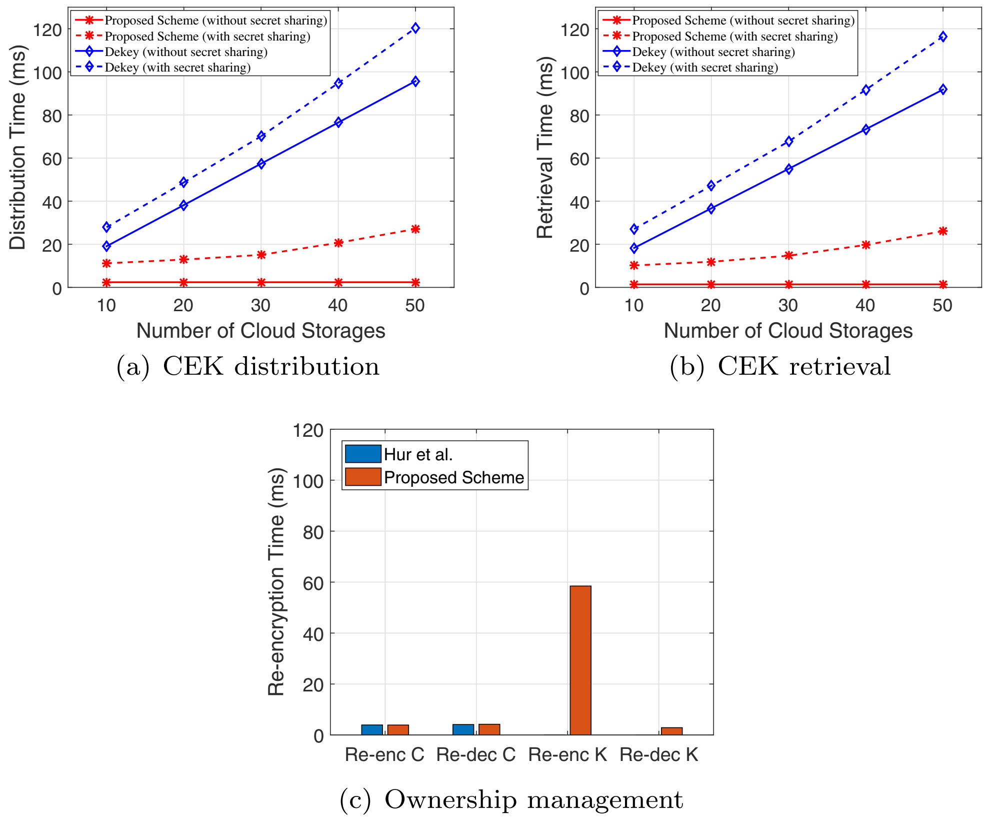


Fig. 5 Communication cost

(a) CEK management with FDs = 5 (b) CEK management with FDs = 15

(c) Ownership management

Fig. 6 Computation cost



is more efficient and scalable as the number of the distributed FDs and the universe of users increase.

5.3.3 Implementation

In order to analyze the computation costs on the prototype, we implemented a CEK and ownership management system. We deployed PBC library version 0.5.14 [27] to compute the three key components using pairing-based cryptography and the EVP library of OpenSSL version 1.0.2c [26] to compute the tag using SHA-256 and to build the secure channels (in Dekey) using RSA encryption, respectively. We implemented the secret sharing scheme using Jerasure version 1.2 [28] as an erasure coding library for the C language. We assigned the Cauchy matrix to the erasure code type and established the parameters n, k, r of the secret sharing scheme as follows: n is the number of FDs, k is half of n, and r is the default value of $k - 1$. For ownership management, we leveraged the CP-ABE open source library [33] with a Type-A curve parameter for privilege-based encryption/decryption and AES-128 in CTR mode for symmetric encryption/decryption. We conducted experiments using a Linux 3.13.0–24 generic OS (64bit) on VMware Player version 6.0.3 [29]. The CPU was an Intel core i7-4500U (1.80GHz) with 1GB RAM.

When the number of FDs increases from 10 to 50 in increments of 10, Fig. 6a and b indicate the key management performance for the CEK distribution and retrieval of Dekey, and the proposed scheme including and excluding the secret sharing scheme. As shown in Table 2, without the secret sharing phase, our scheme exhibits fixed computational costs even if the number of FDs increases. By contrast, that of Dekey linearly increases as the number of FDs increases. Thus, although the secret sharing phase is combined, the overall performance results show that our scheme presents more scalable and efficient CEK management than Dekey regardless of the number of FDs while preserving the security of the key components without establishing a secure channel. Next, Fig. 6c indicates the ownership management performance of Hur et al.'s scheme [11] and the proposed scheme for re-encryption/decryption of ciphertext (Re-enc C and RE-dec C) and key (Re-enc K and Re-dec K). Even if the cost of key re-encryption of the proposed scheme is higher than that of Hur et al.'s scheme (ours: 58.461 ms, Hur et al.'s scheme: 0.024 ms), this is an inevitable trade-off to reduce the communication cost on the client-side.

6 Conclusions

We presented a secure deduplication scheme for fog computing that features reliable and scalable CEK management and provides dynamic ownership management. We divide the CEK into three key components, a pairing key, shared secret, and partial key, and we improved both the reliability and scalability. By employing pairing-based cryptography, we allowed the three key components to be sent via public channels while preserving their privacy. By adopting a re-encryption technique using privilege-based encryption, we reduced the communication costs on the user side when the number of users in the system dynamically changes. The results of a security and performance analysis demonstrate that our scheme is more secure and scalable than previous schemes.

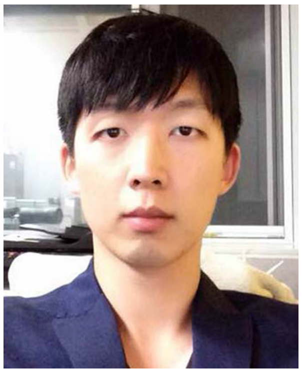
Acknowledgements This work was supported by Institute for Information & communications Technology Promotion(IITP) grant funded by the Korea government(MSIT) (No.2018-0-00269, A research on safe and convenient big data processing methods) (No.2017-0-00184, Self-Learning Cyber Immune Technology Development) (2014-0-00065, Resilient Cyber-Physical Systems Research). This work was also supported by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MSIP) (No. 2016R1A2A2A05005402).

References

1. Bonomi F, Milito R, Zhu J, Addepalli S (2012) Fog computing and its role in the internet of things. In: Proceedings of the first edition of the MCC workshop on mobile cloud computing
2. Stojmenovic I, Wen S (2014) The fog computing paradigm: scenarios and security issues. In: 2014 federated conference on computer science and information systems (FedCSIS)
3. Kraemer FA, Braten AE, Tamkittikhun N, Palma D (2017) Fog computing in healthcare—a review and discussion. IEEE Access
4. Clements AT, Ahmad I, Vilayannur M, Li J et al (2009) Decentralized Deduplication in SAN Cluster File Systems. In: USENIX annual technical conference
5. Douceur JR, Adya A, Bolosky WJ, Simon P, Theimer M (2002) Reclaiming space from duplicate files in a Serverless distributed file system. In: 22nd international conference on distributed computing systems, 2002. Proceedings
6. Bolosky WJ, Douceur JR, Ely D, Theimer M (2000) Feasibility of a serverless distributed file system deployed on an existing set of desktop PCs. ACM SIGMETRICS Performance Evaluation Review 28(1):34–43
7. Dropbox. https://www.dropbox.com. Accessed 13 March 2018
8. Google Drive. https://drive.google.com. Accessed March 13, 2018
9. Mozy. https://www.mozy.com. Accessed March 13, 2018
10. Bellare M, Keelveedhi S, Ristenpart T (2013) Message-locked encryption and secure deduplication. In: Annual international conference on the theory and applications of cryptographic techniques
11. Hur J, Koo D, Shin Y, Kang K (2016) Secure data deduplication with dynamic ownership management in cloud storage. IEEE Trans Knowl Data Eng 28(11):3113–3125
12. Li J, Li YK, Chen X, Lee PPC, Lou W (2015) A hybrid cloud approach for secure authorized deduplication. IEEE Trans Parallel Distrib Syst 26(5):1206–1216
13. Li J, Chen X, Huang X, Tang S, Xiang Y, Hassan MM, Alelaiwi A (2015) Secure distributed deduplication systems with improved reliability. IEEE Trans Comput 64(12):3569–3579
14. Stanek J, Sorniotti A, Androulaki E, Kencl L (2014) A secure data deduplication scheme for cloud storage. In: International conference on financial cryptography and data security
15. Keelveedhi S, Bellare M, Ristenpart T (2013) DupLESS: server-aided encryption for deduplicated storage. Presented as part of the 22nd USENIX Security Symp
16. Duan Y (2014) Distributed key generation for encrypted deduplication: achieving the strongest privacy. In: Proceedings of the 6th edition of the ACM workshop on cloud computing security
17. Li J, Chen X, Li M, Li J, Lee PPC, Lou W (2014), Secure deduplication with efficient and reliable convergent key management. IEEE Trans Parallel Distrib Syst 25(6):1615–1625
18. Blakley GR, Meadows C (1984) Security of ramp schemes. In: Workshop on the theory and application of cryptographic technique
19. Bellare M, Namprempre C, Pointcheval D, Semanko M (2003), The one-more-RSA-inversion problems and the security of Chaum's blind signature scheme. J Cryptol 16(3):185–215
20. Li M, Qin C, Lee PPC, Li J (2014) Convergent dispersal: toward storage-efficient security in a Cloud-of-Clouds. HotCloud
21. Li M, Qin C, Li J, Lee PPC (2016) Cdstore: toward reliable, secure, and cost-efficient cloud storage via convergent dispersal. IEEE Internet Comput 20(3):45–53
22. Halevi S, Harnik D, Pinkas B, Shulman-Peleg A (2011). In: Proceedings of the 18th ACM conference on computer and communications security. Proofs of ownership in remote storage systems
23. Laurie B, Langley A, Kasper E (2013) Certificate transparency. IETF
24. Fahl S, Harbach M, Muders T, Smith M (2012) Confidentiality as a service–usable security for the cloud. In: 2012 IEEE 11th international conference on trust, security and privacy in computing and communications (TrustCom)
25. Fahl S, Harbach M, Muders T, Smith M, Sander U (2012) Helping Johnny 2.0 to Encrypt His Facebook conversations. In: Proceedings of the eighth symposium on usable privacy and security
26. OpenSSL Project. https://www.openssl.org. Accessed 13 March 2018
27. The pairing-based cryptography library. https://crypto.stanford.edu/pbc/. Accessed 13 March 2018
28. Plank JS, Simmerman S, Schuman CD (2008) Jerasure: a library in C/C++ facilitating erasure coding for storage applications-version 1.2. Citeseer
29. VMware. https://www.vmware.com. Accessed 13 March 2018
30. Shin Y, Koo D, Hur J (2017) A survey of secure data deduplication schemes for cloud storage systems. In: ACM computing surveys (CSUR)
31. Kwon H, Hahn C, Koo D, Hur J (2017) Scalable and reliable key management for secure deduplication in cloud storage. In: 2017 IEEE 10th international conference on cloud computing (CLOUD)
32. Jiang S, Jiang T, Wang L (2017) Secure and efficient cloud data deduplication with ownership management. IEEE Trans Serv Comput PP(99):1-1
33. Bethencourt J, Sahai A, Waters B (2007) Ciphertext-policy attribute-based encryption. In: IEEE symposium on security and privacy, 2007. SP'07
34. Goyal V, Pandey O, Sahai A, Waters B (2006) Attribute-based encryption for fine-grained access control of encrypted data. In: Proceedings of the 13th ACM conference on computer and communications security

35. Sahai A, Waters B (2005) Fuzzy identity-based encryption. In: Annual international conference on the theory and applications of cryptographic techniques
36. Shamir A (1984) Identity-based cryptosystems and signature schemes. In: Workshop on the theory and application of cryptographic techniques
37. Russell A, Wang H (2002) How to fool an unbounded adversary with a short key. In: International conference on the theory and applications of cryptographic techniques

Hyunsoo Kwon received the B.S. degree of computer science and engineering from Chung-Ang University, Seoul, South Korea, in 2014, and the M.S. degrees from Korea University, Seoul in 2016. He is currently pursuing the Ph.D. degree in the Department of Computer Science and Engineering, Korea University, Seoul, 156-756, South Korea. His research interests include information security, mobile computing security, network security, and applied cryptography.



Changhee Hahn received the B.S. and M.S. degrees from Chung-Ang University, Seoul, South Korea, in 2014 and 2016, respectively, all in Computer Science. He is currently pursuing the Ph.D. degree in the Department of Computer Science and Engineering, Korea University, Seoul, 156-756, South Korea. His research interests include information security, mobile computing security, cyber security, and applied cryptography.

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 UrbanaChampaign. In 2011, he was with the Department of Computer Science and Engineering, Hanyang University, Ansan, 426-791, 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 cyberphysical systems. He is a member of the ACM.

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 Associate Professor with the Department of Computer Science and Engineering at the Korea University, Seoul, 156-756, South Korea. His research interests include information security, cloud computing security, mobile security, and applied cryptography.