IEICE TRANS. INF. & SYST., VOL.E97–D, NO.2 FEBRUARY 2014

PAPER

Efficient and Secure File Deduplication in Cloud Storage

Youngjoo SHIN[†,††a)] ***and*** **Kwangjo KIM**[†b)]**,** ***Members***

SUMMARY Outsourcing to a cloud storage brings forth new challenges for the efficient utilization of computing resources as well as simultaneously maintaining privacy and security for the outsourced data. Data deduplication refers to a technique that eliminates redundant data on the storage and the network, and is considered to be one of the most-promising technologies that offers efficient resource utilization in the cloud computing. In terms of data security, however, deduplication obstructs applying encryption on the outsourced data and even causes a side channel through which information can be leaked. Achieving both efficient resource utilization and data security still remains open. This paper addresses this challenging issue and proposes a novel solution that enables data deduplication while also providing the required data security and privacy. We achieve this goal by constructing and utilizing equality predicate encryption schemes which allow to know only equivalence relations between encrypted data. We also utilize a hybrid approach for data deduplication to prevent information leakage due to the side channel. The performance and security analyses indicate that the proposed scheme is efficient to securely manage the outsourced data in the cloud computing.

key words: *cloud computing security, data deduplication, predicate encryption, online guessing attack*

1. Introduction

Cloud computing is a promising technology that economically enables data outsourcing as a service using Internet technologies with elastic provisioning and usage-based pricing [1]. As demand for data outsourcing increases, pay-as-you-use cloud paradigm drives the need for cost-efficient storage, specifically for reducing storage space and network bandwidth overhead, which is directly related to the financial cost savings. Furthermore, the need for data security and privacy also arises as more and more sensitive data are being outsourced to cloud, such as emails and personal health records, *etc*. For the fast growth of the cloud computing, both problems of cost efficiency and data security should be addressed well and resolved simultaneously.

In order to achieve cost savings, commercial service providers utilize their resources efficiently through cross-user data deduplication [2], which refers to a technique that eliminates copies of a redundant file (or data chunk) across users and provides links to the file instead of storing the copies. There are two approaches of deduplication based on its architecture. In the server-side (*i.e.*, target-based) deduplication, the cloud storage server mainly handles deduplication. On the other hand, client-side (*i.e.*, source-based) deduplication occurs at the client before a file is transferred to the server. Client-side deduplication improves not only storage space but also network bandwidth utilization. As noted in [3], this technique achieves disk and bandwidth savings of more than 90%, and thus most commercial storage service providers including DropBox, Mozy and Memopal take advantage of client-side deduplication.

Data security is another challenging issue in cloud computing, which originates from the fact that cloud servers are usually outside of the trust domain of the data owners. For recent years, the security research community has mainly addressed privacy or security on the outsourced cloud data, specifically focusing on access control [4]–[8] and searchability over the encrypted data [9]–[13]. These proposed solutions are usually attained through a sort of encryption techniques. Most commercial storage service providers, however, are reluctant to apply encryption on the stored data, because encrypting on data impedes executing data deduplication [3], [14]. Storage service providers may be unlikely to stop using deduplication due to the high cost savings offered by the technique.

Several approaches [15]–[18] proposed some ideas to enable data deduplication over the encrypted data. Their solutions are commonly based on so-called convergent encryption, which takes a hashed value of a file as an encryption key. Using this technique, identical ciphertexts are always generated from identical plaintexts in a deterministic manner. Hence, the cloud server can perform deduplication over encrypted data, without knowing the exact content of the file. From the view of cryptography, however, it is understood that deterministic encryption, including convergent encryption naturally, is not as secure as randomized one [19].

Keeping confidentiality of the outsourced data from the untrusted cloud server is not the only security problem for the system utilizing data deduplication. It has been shown that most of storage services using client-side deduplication commonly incur a side channel through which an adversary may learn information about the contents of files of other users [3]. This is because these services share two inherent properties; 1) data transmission in a network can be visible to an adversary, and 2) deterministic and small-size data, such as a hashed value of a file, is queried to the cloud server before file uploading. A user who has not access to but curious about some file might mount an online guessing attack

Manuscript received November 19, 2012.
Manuscript revised October 22, 2013.

[†]The authors are with the Department of Computer Science, KAIST, Korea.
[††]The author is with the Attached Institute of ETRI, Korea.
a) E-mail: s.youngjoo@kaist.ac.kr
b) E-mail: kkj@kaist.ac.kr

DOI: 10.1587/transinf.E97.D.184

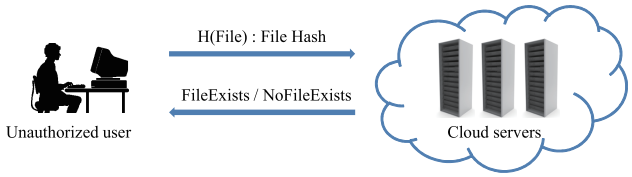


Fig. 1 Online Guessing Attack Scenario: for each possible versions of target file unauthorized user continues querying its hashed value until receiving 'FileExists' message from the cloud server.

exploiting the side channel. The attack is as follows: an adversary first constructs a search domain comprising possible versions of the target file, and then queries a hash of each version to the server until a deduplication event occurs. (See Fig. 1) This attack is very relevant for a corporate environment where the file content usually has low-entropy. That is, most of files are small variations of standard templates and the number of possible versions of the target file is moderate. It is challenging but certainly necessary to prevent such an information leakage in the client-side deduplication system.

To sum up, the two issues of efficient resource utilization and data security in the cloud computing have not been considered together and well addressed yet in either of academia or industries. It actually still remains open to achieve data security against the untrusted server as well as unauthorized users capable of online guessing attack, while enabling data deduplication.

In this paper, we address this issue and propose a novel solution for cloud storage that raises efficiency up to a level of practicality while also achieving strong security. Concretely, we propose a data deduplication scheme which is secure against the untrusted server as well as unauthorized users. Our proposed scheme utilizes primarily a novel cryptographic primitive, namely equality predicate encryption, which allows cloud server to know only equivalence relations among ciphertexts without leaking any other information of their plaintexts. Thus data privacy can be kept from the cloud server while data deduplication is still enabled over the encrypted files. In addition, the proposed scheme allows the cloud server to perform data deduplication in a hybrid manner. That is, deduplication will occurs either of at server side or at client side with some probability. This strategy greatly reduces the risk of information leakage by increasing attack overhead of online-guessing adversaries. In order to achieve our design goals, we constructed two equality predicate encryption schemes in the symmetric-key setting suitable for our application. The proposed scheme is built upon these constructions, and the required data security is strongly enforced through the underlying provable security of the constructions.

Main contributions of this paper can be summarized as follows: 1) This paper is the first one that simultaneously resolved two issues of the data security and the efficient resource utilization in the cloud computing; 2) The proposed scheme also gives data owners a guarantee that adding their data to the cloud storage has a very limited effect on what an adversary exploiting side channel may learn about the data; 3) The proposed scheme as well as our constructed equality predicate encryptions are proved to be secure under the standard cryptographic assumption.

The rest of this paper is organized as follows: Section 2 presents the system models and an overview of our solution. Section 3 describes our constructions of equality predicate encryption scheme. In Sect. 4, we describe the proposed secure data deduplication scheme based on the constructions of the encryption scheme in detail. In Sect. 5, we analyze the security and discuss the performance of the proposed scheme. We describe related works in Sect. 6. Finally, we conclude our work in Sect. 7.

2. Models and Solution Overview

2.1 System and Attack Model

We assume that the system consists of the three entities: Data owners, users and a cloud server. A data owner wishes to outsource data (or file) in the external storage server and is in charge of encrypting the data before outsourcing it. In order to access the outsourced data, a user should be authorized by the data owner. An authorized user possess the secret key of the data file and can read the data by decrypting it. The cloud server, which stores outsourced data from data owners, has abundant but limited storage and network capacity. The cloud server is operated by the cloud service provider who is interested in cost savings by improving disk space and network utilization.

Similar to the previous approaches, we just assume the cloud server to be honest-but-curious. That is, the cloud server will honestly execute the assigned tasks in the system but may try to learn some information of stored data as much as possible. Unauthorized users would try to access stored data which is not authorized to them. To achieve this goal, they may perform an online guessing attack using the side channel incurred by a client-side duplication as an oracle. In addition, users and other data owners may also collude to compute secret information that is necessary to access data with any information that colluders have. Both of the cloud server and any unauthorized users mounting these attacks should be prevented from getting any information of unauthorized data on the storage.

We also consider malicious data owners who mount some kind of attacks targeting a cloud server or other data owners. In uploading a file, they would try to disturb deduplication or corrupt data owned by others by doing dishonest behaviors such as computing with a manipulated secret key and uploading a fake file. Reliability of the proposed scheme against such attacks should be retained as well.

2.2 Solution Overview

Our goal is to help the cloud server enjoy the cost savings offered by data deduplication, while also giving data owners a guarantee that their data is kept confidential against the

untrusted cloud server and the unauthorized users. Specifically, our motivation is to design an efficient and secure cross-user data deduplication scheme for cloud storage services. For this, we have primarily constructed, based on PEKS (Public-key Encryption with Keyword Search) [9], two equality predicate encryptions schemes in the symmetric-key setting: 1) SEPE (Symmetric-key Equality Predicate Encryption) which is a basic equality predicate encryption scheme and 2) SEPE$_n$ which is an extended scheme of SEPE that has one-to-n tokens mapping ($n \geq 1$). That is, SEPE$_n$ allows n possible tokens to be matched the corresponding encrypted file. Our constructions allow the cloud server in possession of a token associated with each file to know the equivalence relations between the encrypted files without knowledge of the file content.

In the proposed scheme, a data owner generates ciphertexts of a file and the corresponding tokens using both of SEPE and SEPE$_n$ before uploading the file to the cloud server. Then the cloud server retrieves an identical file in the storage by comparing these tokens with stored ciphertexts. If an identical file has existed in the storage, using SEPE, the cloud server will always perform server-side deduplication even without knowing the file content. On the other hand, SEPE$_n$ will enables client-side deduplication with some probability less than 1. This is because just one out of n possible tokens will be correctly verified with the stored file. Hence, using SEPE$_n$ gives online guessing adversaries substantial search complexity so that no adversary with probabilistic and polynomially bounded computational resources can recover the information. With help of SEPE and SEPE$_n$, the proposed scheme provides data security while also enabling deduplication either of at server-side or at client-side in a hybrid manner. Some network transmission overhead may be introduced due to the randomized occurrence of client-side deduplication, but the overhead can be minimized while ensuring the required security as we will discuss later.

3. Symmetric-Key Equality Predicate Encryption

In this section, we formally define symmetric-key equality predicate encryption and its security requirement. Then, we construct two encryption schemes, SEPE and SEPE$_n$, which fulfill the definition and security requirement. The security of the constructed schemes is also proved under the cryptographic complexity assumptions rigorously.

3.1 Definitions

3.1.1 Symmetric-Key Equality Predicate Encryption

We first describe the concept of predicate encryption before giving the definition of symmetric-key equality predicate encryption. By definitions in [20] and [21], predicate encryption is a kind of functional encryption system in which a token associated with a function allows a user to evaluate the function over the encrypted data. In predicate encryption scheme, an encryption of a plaintext M can be evaluated using a token associated with a predicate to learn whether M satisfies the predicate. Symmetric-key equality predicate encryption is a predicate encryption scheme in the symmetric-key setting that allows to evaluate an equality predicate over a ciphertext of M given a token of another plaintext M' to learn whether M and M' are equal. Symmetric-key setting means that both of encryption and token generation are computed with the same secret key. Note that PEKS [9] is an equality predicate encryption scheme in the public-key setting.

Definition 1. *A symmetric-key equality predicate encryption scheme consists of the following probabilistic and polynomial time algorithms:*
Initialize(1^λ): Take as input a security parameter 1^λ and output a global parameter Param*. For brevity, the global parameter* Param *output by Initialize algorithm is omitted below.*
KeyGen(aux): Output a secret key SK. An auxiliary input aux may be used in generating the secret key.
Encrypt(SK, M): Take as input a secret key SK and a plaintext M and output a ciphertext CT.
GenToken(SK, M): Take as input a secret key SK and a plaintext M and output a token TK.
Test(TK, CT): Take as input TK=GenToken(SK, M') and CT=Encrypt(SK, M), and output 'Yes' if $M = M'$ and 'No' otherwise.

3.1.2 Security Definition

The required security of symmetric-key equality predicate encryption scheme is an adaptive chosen plaintext security. We define the security against an active adversary who can access an encryption oracle and also obtain tokens TK for any plaintext M of his choice. Even under such an attack, the adversary is not allowed to distinguish an encryption of a plaintext M_0 from an encryption of a plaintext M_1. Now we define the security game between a challenger and the adversary $\mathcal{A}$, as below:

Setup: Challenger runs the *Initialize* algorithm to generate a global parameter Param and the *KeyGen* algorithm to generate a secret key SK. It gives the global parameter to the adversary and keeps SK to itself.
Phase 1: The adversary can adaptively ask the challenger for ciphertext CT or token TK for any plaintext $M \in \{0,1\}^*$ of his choice.
Challenge: Once $\mathcal{A}$ decides that Phase 1 is over, $\mathcal{A}$ sends two challenging plaintexts M_0, M_1 to the challenger. The restriction is that $\mathcal{A}$ did not previously ask for the ciphertexts or the tokens of M_0 and M_1. The challenger picks a random $b \in \{0,1\}$ and gives $\mathcal{A}$ a CT of M_b.
Phase 2: $\mathcal{A}$ continues to ask for ciphertexts CT or tokens TK for any plaintext M of his choice except M_0, M_1.
Guess: Finally, $\mathcal{A}$ outputs $b' \in \{0,1\}$ and wins the game if $b = b'$.

We define the adversary $\mathcal{A}$'s advantage in breaking the symmetric-key equality predicate encryption scheme as

$$\mathrm{Adv}_{\mathcal{A}} = \left|\Pr\{b = b'\} - \frac{1}{2}\right|.$$

Definition 2. *A symmetric-key equality predicate encryption scheme is semantically secure against an adaptive chosen plaintext attack if for any probabilistic and polynomial time (PPT) adversary $\mathcal{A}$, the advantage of $\mathcal{A}$ in winning the game is negligible in λ.*

3.2 Cryptographic Background

In this section, we briefly review the bilinear maps and its complexity assumption on which our constructions are built.

3.2.1 Bilinear Maps

Our constructions are based on some facts about groups with efficiently computable bilinear maps. Let $\mathbb{G}$ and $\mathbb{G}_T$ be two multiplicative cyclic groups of prime order p. Let g be a generator of $\mathbb{G}$. A bilinear map is an injective function $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ with following properties:

1. *Bilinearity*: for all $u, v \in \mathbb{G}$ and $a, b \in \mathbb{Z}_p^*$, we have $e(u^a, v^b) = e(u, v)^{ab}$.
2. *Non-degeneracy*: $e(g, g) \neq 1$.
3. *Computability*: There is an efficient algorithm to compute $e(u, v)$ for $\forall u, v \in \mathbb{G}$.

3.2.2 Bilinear Diffie-Hellman (BDH) Assumption

Let $a, b, c \in \mathbb{Z}_p^*$ be chosen at random and g be a generator of $\mathbb{G}$. The Bilinear Diffie-Hellman problem is to compute $e(g, g)^{abc} \in \mathbb{G}_T$ given $g, g^a, g^b, g^c \in \mathbb{G}$ as input. The BDH assumption [22], [23] states that no PPT algorithm can solve the BDH problem with non-negligible advantage.

3.3 Our Constructions

We give two constructions for symmetric-key equality predicate encryption scheme based on Bilinear Diffie-Hellman assumption in the random oracle model: (1) SEPE which is a basic construction of symmetric-key equality predicate encryption scheme and (2) SEPE$_n$ which has one-to-n tokens mapping so that n multiple tokens are possible to the corresponding ciphertext. We will need the following hash functions $H_0 : \{0,1\}^* \to \mathbb{Z}_p^*$, $H_1 : \{0,1\}^* \to \mathbb{G}$, $H_2^i : \{0,1\}^* \to \mathbb{Z}_p^*$, where $i \in \mathbb{N}$ is an index and $H_3 : \mathbb{G}_T \to \{0,1\}^{\log p}$. H_2^i can be easily constructed from a keyed hash algorithm like MAC in which an index i is used as a key.

3.3.1 SEPE

SEPE scheme consists of the following algorithms.

Initialize(1^λ): The initialize algorithm randomly chooses a prime p with the bit length λ and generates a bilinear group $\mathbb{G}$ of order p with its generator g. Then, it outputs a tuple $\mathsf{Param}=\langle p, g, \mathbb{G}, \mathbb{G}_T\rangle$ as a global parameter. Note that Param is taken as an input implicitly next algorithms.
KeyGen(aux): The key generating algorithm takes a binary string aux with arbitrary length as an auxiliary input and computes $SK = H_0(aux) \in \mathbb{Z}_p^*$. Then, it outputs SK as a secret key.
Encrypt(SK, M): Let us denote $SK{=}\alpha \in \mathbb{Z}_p^*$. The encryption algorithm first chooses $r \in \mathbb{Z}_p^*$ at random and then computes a ciphertext $CT = \langle h^r, g^r, H_3(e(h, g^\alpha)^r)\rangle$, where $h = H_1(M) \in \mathbb{G}$.
GenToken(SK, M): Let us denote $SK{=}\alpha \in \mathbb{Z}_p^*$. The algorithm first chooses $k \in \mathbb{Z}_p^*$ at random and then computes a token of a plaintext M as $TK = \langle h^{\alpha+k}, g^k\rangle$, where $h = H_1(M) \in \mathbb{G}$.
Test(TK, CT): We assume that $CT = \langle c_1, c_2, c_3\rangle$ for a plaintext M and $TK = \langle t_1, t_2\rangle$ for a plaintext M'. The test algorithm computes

$$\gamma = e(t_1, c_2)/e(c_1, t_2).$$

If $c_3 = H_3(\gamma)$, it outputs 'Yes'; if not, it outputs 'No'.

The algorithm *KeyGen* generates a secret key from *aux* which is distributed over $\{0,1\}^*$. Let us denote the min-entropy of the distribution of *aux* by H_∞^{aux}.

Theorem 1. *SEPE described above is semantically secure against an adaptively chosen plaintext attack in the random oracle model assuming that BDH is intractable and H_∞^{aux} is a polynomial in λ.*

Proof. Refer to Appendix A for the proof. □

3.3.2 SEPE$_n$

SEPE$_n$ scheme consists of the following algorithms.

Initialize$_n(1^{\lambda'})$: The initialize algorithm parses an input $1^{\lambda'}$ in a polynomial time as a tuple $\langle 1^\lambda, d\rangle$ that contains two security parameters 1^λ and d. The algorithm randomly chooses a prime p with the bit length λ and generates $\mathbb{G}$ of order p with its generator g. Then, it outputs $\mathsf{Param}=\langle p, g, \mathbb{G}, \mathbb{G}_T, d\rangle$ as a global parameter. Note that the security parameter d inherently defines a set of permutations Φ in which a permutation domain is $D = \{1, 2, \ldots, d\} \subset \mathbb{N}$.
KeyGen$_n(aux)$: The key generating algorithm takes a binary string aux with arbitrary length as an auxiliary input and outputs a secret key $SK{=}\langle\alpha, \pi\rangle$ by computing $\alpha = H_0(aux) \in \mathbb{Z}_p^*$ and choosing randomly π from Φ.
Encrypt$_n(SK, M)$: Let us denote $SK{=}\langle\alpha, \pi\rangle$, where $\alpha \in \mathbb{Z}_p^*$ and $\pi : D \to D \in \Phi$. The encrypt algorithm first chooses $r \in \mathbb{Z}_p^*$ at random and then computes $h \in \mathbb{G}$ by using two vectors $\vec{u} = (1, 2, \ldots, d)$ and $\vec{v}_\pi = (H_2^{\pi(1)}(M), H_2^{\pi(2)}(M), \ldots, H_2^{\pi(d)}(M))$,

$$h = g^{\vec{u}\cdot\vec{v}_\pi}, \text{ where } \vec{u}\cdot\vec{v}_\pi = \sum_{i\in D} i \cdot H_2^{\pi(i)}(M).$$

Table 1 Comparison of predicate encryption schemes.

	PEKS [9]	Shen *et al.* [21]	Blundo *et al.* [24]	SEPE (SEPE$_n$)
Secret key environment	public-key	symmetric-key	symmetric-key	symmetric-key
Predicate type	equality	multiple	multiple	equality
Cross-user deduplication	No	Yes	Yes	Yes
One-to-n token mapping	No	No	No	Yes
Ciphertext size	$\|\mathbb{G}\| + \log g$	$\geq 4\|\mathbb{G}\|$	$\geq 2\|\mathbb{G}\| + \|\mathbb{G}_T\|$	$2\|\mathbb{G}\| + \log g$
Token size	$\|\mathbb{G}\|$	$\geq 4\|\mathbb{G}\|$	$\geq 2\|\mathbb{G}\|$	$2\|\mathbb{G}\|$
Encrypt algorithm cost	2e+p	$\geq$ 8m + 8e	$\geq$ 3e	3e+p
GenToken algorithm cost	e	$\geq$ 8m + 8e	$\geq$ 2e	2e
Test algorithm cost	p	$\geq$ 3m + 4p	$\geq$ 2m + 2p	m+2p

$|\mathbb{G}|$: size of an element in $\mathbb{G}$, $|\mathbb{G}_T|$: size of an element in $\mathbb{G}_T$, g: order of $\mathbb{G}$, m: multiplication (division), e: exponentiation, p: pairing

The ciphertext is $CT = \langle h^r, g^r, H_3(e(h, g^\alpha)^r)\rangle$.

$GenToken_n(SK, M)$: Let us denote $SK{=}\langle\alpha, \pi\rangle$. The algorithm first chooses $k \in \mathbb{Z}_p^*$ at random and then computes $h = g^{\vec{u}\cdot\vec{v}_\pi} = g^{\sum_{i\in D} i\cdot H_2^{\pi(i)}(M)}$. The token of the plaintext M is $TK = \langle h^{k+\alpha}, g^k\rangle$.

$Test_n(TK, CT)$: We assume that $CT = \langle c_1, c_2, c_3\rangle$ for a plaintext M and $TK = \langle t_1, t_2\rangle$ for a plaintext M'. The test algorithm computes

$$\gamma = e(t_1, c_2)/e(c_1, t_2).$$

If $c_3 = H_3(\gamma)$, it outputs 'Yes'; if not, it outputs 'No'.

Theorem 2. *SEPE$_n$ described above is semantically secure against an adaptively chosen plaintext attack in the random oracle model assuming that BDH is intractable and H_∞^{aux} is a polynomial in λ.*

Proof. Refer to Appendix B for the proof. □

3.4 Discussion

We discuss the effectiveness of our constructions, SEPE and SEPE$_n$, by comparing with several predicate encryption schemes that are similar to ours. Table 1 summarizes the comparisons of predicate encryption schemes.

PEKS [9] is a public-key equality predicate encryption scheme that allows to identify equality relations among ciphertexts and tokens. PEKS is designed to be run in the public-key environment, and thus the equality of ciphertexts that are encrypted with one's public key can only be tested with tokens that are generated with the corresponding private key. This public-key setting is not suitable for cross-user data deduplication, in which the equality test should be performed across ciphertexts and tokens that are generated with different users' secret keys. For cloud storage services, it is more desirable to perform data deduplication across multiple users' outsourced data than deduplicating only over a user's data, since it gives more chance of eliminating the redundancy and saves more resources of the cloud server.

Cross-user deduplication can be implemented by a symmetric-key type of predicate encryption algorithm. Shen *et al.* [21] and Blundo *et al.* [24] proposed symmetric-key predicate encryption schemes which support more general predicates such as a comparison predicate ($x \geq a$), a subset predicate ($x \in S$), and arbitrary conjunctive predicates ($P_1 \wedge P_2 \wedge \ldots \wedge P_l$). Both schemes also support predicate privacy, which is a security property that a token reveals no information about the predicate that it contains. For data deduplication that only the equality predicate is sufficient, however, the feature that supports multiple predicates is not necessary, and is inefficient in terms of ciphertext size (token size) and computation time of encryption, token generation and predicate testing. Predicate privacy is also unnecessary for data deduplication, since only one type of predicate is required.

SEPE and SEPE$_n$ are designed to be more suitable for data deduplication. The symmetric-key based construction makes it applicable to cross-user deduplication that eliminates redundancy of data across multiple users. Moreover, SEPE (SEPE$_n$) uses less storage resources (*i.e.,* size of ciphertexts and tokens) and incurs less computation, particularly in *Test* algorithm that causes most of the computational burden in the cloud server, compared with other symmetric-key predicate encryption schemes. One-to-n token mapping of SEPE$_n$ is another feature that makes our constructions more useful for designing secure data deduplication.

4. The Proposed Data Deduplication Scheme

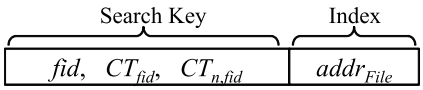
4.1 Definition and Notation

We state our definitions and notations. Table 2 gives the description of notations to be used in the proposed scheme. For each file, a data owner assigns a globally unique file id fid and encrypts with a set of randomly chosen symmetric encryption keys FEK={FEK$_1$,FEK$_2$,...,FEK$_l$}. Also, FEK is encrypted with another secret key α. The cloud server maintains a search index $\mathcal{SI}$ to retrieve a stored file in the storage. $\mathcal{SI}$ is a list of entries in which each entry corresponds to a file entity logically, and comprises a set of search keys and an index value. Search keys include a file id fid and two predicate ciphertexts CT_{fid}, $CT_{n,fid}$ which are encrypted with SEPE and SEPE$_n$, respectively. Files in the storage are retrieved by the cloud server with one of search keys.

The usage of a search key depends on a type of file operation requested by users. An index value contains an address $addr_{File}$ at which the file is physically stored in the system. Each $\mathcal{SI}$ entry has its address $addr_{Index}$ and can be

Table 2 Notations used in the proposed scheme description.

Notation	Description
$\text{FEK}_1, \ldots, \text{FEK}_l$	symmetric file encryption keys
$\mathcal{SI}$	search index for file retrieval in the cloud storage
fid	a unique id assigned to each file
$addr_{obj}$	address at which an object obj resides in the system
CT_{fid}	SEPE ciphertext of a file of fid
$CT_{n,fid}$	SEPE_n ciphertext of a file of fid
$CT_{fid}^{[i]}$	i-th element of CT_{fid} (i=1,2,3)
$CT_{n,fid}^{[i]}$	i-th element of $CT_{n,fid}$ (i=1,2,3)
TK_{fid}	token which corresponds to ciphertext CT_{fid}
$TK_{n,fid}$	token which corresponds to ciphertext $CT_{n,fid}$
$TK_{fid}^{[i]}$	i-th element of TK_{fid} (i=1,2)
$TK_{n,fid}^{[i]}$	i-th element of $TK_{n,fid}$ (i=1,2)
$\{M\}_{key}$	ciphertext of M encrypted by a symmetric encryption algorithm with an encryption key key
$<i_1,i_2,\ldots,i_n>$	tuple of n elements $i_1, i_2, \ldots, i_n$



(a) Format of an entry in the search index

(b) Structure of a file storage

Fig. 2 A search index and a file storage on a cloud server.

referenced through the address. Figure 2 shows the format of a $\mathcal{SI}$ entry and the structure of file storage.

For data reliability, an erasure code is used in the proposed scheme. An erasure (x, y)-code is an encoding function $\mathcal{E} : X \rightarrow Y$ that transforms a message X comprised of x symbols to the code Y of $x + y$ redundant symbols such that any x of them are sufficient to reconstruct the original message X. The number of losses that the erasure code $\mathcal{E}$ can sustain is y, while the redundancy factor is $\mathsf{RD} = (x + y)/x$.

4.2 Scheme Description

The proposed scheme is composed of several system level operations: *System Setup*, *New File Upload*, *File Access*, *File Update* and *File Deletion*.

System Setup: Initially, the cloud server and data owners have agreed two security parameters λ and d. The cloud server calls the algorithm $Initialize_n(\langle\lambda, d\rangle)$ and outputs the global parameter $\mathsf{Param}=\langle p, g, \mathbb{G}, \mathbb{G}_T, d\rangle$. The parameters p, g, $\mathbb{G}$ and $\mathbb{G}_T$ will be used commonly in both of SEPE and SEPE_n.

New File Upload: When uploading a file to the cloud server, a data owner proceeds as the following:

1. assign a unique file id fid to this data file.
2. generate a secret key $\langle\alpha, \pi\rangle$ by running $KeyGen_n(File)$, where $File$ is a binary representation of this file content. (α is also assigned to the secret key of SEPE.)
3. compute two tokens TK_{fid} and $TK_{n,fid}$ by running $GenToken(\alpha, File)$ and $GenToken_n(\langle\alpha, \pi\rangle, File)$, respectively.
4. compute two ciphertexts CT_{fid}=$Encrypt(\alpha, File)$ and $CT_{n,fid}$=$Encrypt_n(\langle\alpha, \pi\rangle, File)$.
5. query to the cloud server with a tuple $<fid, TK_{fid}, TK_{n,fid}, CT_{fid}, CT_{n,fid}>$.

To ensure that the tuple is correct, the data owner is requested to compute ciphertexts and tokens by sharing random values $r, k \in \mathbb{Z}_p^*$ between SEPE and SEPE_n at steps 3 and 4 above. (Thus, the tuple should satisfy $CT_{fid}^{[2]} = CT_{n,fid}^{[2]}$ and $TK_{fid}^{[2]} = TK_{n,fid}^{[2]}$.)

(Tuple verification) Upon receiving the tuple, the cloud server first verifies the consistency of a received tuple by checking both $Test_n(TK_{n,fid}, CT_{n,fid})$ and $Test(TK_{fid}, CT_{fid})$ output 'Yes'. Then, the cloud server continues to test the Eqs. (1), (2) and (3).

$$e\left(CT_{fid}^{[1]}, TK_{n,fid}^{[1]}\right) = e\left(CT_{n,fid}^{[1]}, TK_{fid}^{[1]}\right) \quad (1)$$

$$CT_{fid}^{[2]} = CT_{n,fid}^{[2]} \quad (2)$$

$$TK_{fid}^{[2]} = TK_{n,fid}^{[2]}. \quad (3)$$

If one of the verification processes fails, the cloud server reports an error and halts the operation.

(Deduplication) If the tuple is successfully verified, the cloud server retrieves entries in the search index $\mathcal{SI}$ comparing $TK_{n,fid}$ with $CT_{n,fid'}$ of each $\mathcal{SI}$ entry through the algorithm $Test_n(TK_{n,fid}, CT_{n,fid'})$. If a matching entry found, the cloud server gets an address $addr_{File}$ from the entry and creates a new entry $\langle fid, CT_{fid}, CT_{n,fid}, addr_{File}\rangle$. Then the cloud server appends it to $\mathcal{SI}$ and responds the data owner with a message $FileExist$ (*i.e.,* client-side deduplication event occurs). If not, the cloud server responds with a message $FileNotExist$.

Upon receiving a message $FileNotExist$, the data owner continues to upload the actual file to the cloud server as the following:

1. transform $File$ into $\mathcal{F}$ by running the erasure code $\mathcal{E}$, and split $\mathcal{F}$ according to block length b (*e.g.,* b=512 bits) into l blocks $\mathcal{F}_1, \mathcal{F}_2, \ldots, \mathcal{F}_l$ (padding is added if the length of $\mathcal{F}$ is not aligned to b).
2. choose randomly encryption keys $\text{FEK}_1, \text{FEK}_2, \ldots, \text{FEK}_l$ from the key space.
3. encrypt the file blocks with a symmetric encryption algorithm $C = \langle\{\mathcal{F}_1\}_{\text{FEK}_1}, \{\text{FEK}_1\}_\alpha, \ldots, \{\mathcal{F}_l\}_{\text{FEK}_l}, \{\text{FEK}_l\}_\alpha\rangle$, where α is a hashed value of the file described above.
4. send a tuple $<fid, C>$ to the cloud server.

Upon receiving the message $<fid, C>$ from the data owner, the cloud server finds the same file comparing TK_{fid} with $CT_{fid'}$ of each $\mathcal{SI}$ entry through $Test(TK_{fid}, CT_{fid'})$.

If the same file already exists in the storage, the cloud server randomly picks l_r ($0 \leq l_r \leq l$), and replaces l_r arbitrary blocks of the existing file (including their FEKs) by

newly uploaded l_r blocks from C.

The cloud server then creates new entry $\langle fid, CT_{fid}, CT_{n,fid}, addr_{File}\rangle$, where $addr_{File}$ is from the matching entry and appends it to $\mathcal{SI}$ (*i.e.,* server-side deduplication event occurs). If not found, the cloud server allocates new storage space with new address $addr_{File}$ and puts the encrypted file C into the file system. Then the server creates new $\mathcal{SI}$ entry $\langle fid, CT_{fid}, CT_{n,fid}, addr_{File}\rangle$ and appends it to $\mathcal{SI}$. Instead of two distinct searches in this operation, it is possible to perform a single search during uploading by retrieving $\mathcal{SI}$ with CT_{fid} and $CT_{n,fid}$ at once.

As described above, the hashed value α acts not only as a plaintext of SEPE and SEPE$_n$ but also as an encryption key of FEK associated with the file itself. Using α as an encryption key helps keeping the system simple, since data owners in possession of same file should also exclusively share a hashed value of the file without any explicit key sharing. If a random encryption key is used instead of the file hash, then we will need costly key management mechanism such as broadcast encryption [25] to share the key among data owners who have a common file. Such a strategy is similar to convergent encryptions, but not the heart of the technique that enables deduplication contrary to convergent encryptions.

File Access: A user authorized to access a file stored in a cloud storage should be in possession of $\langle fid, \alpha\rangle$ of the file. The user sends a file access request message with a file id fid. The cloud server finds the stored file retrieving the search index $\mathcal{SI}$ with a search key fid. If found, the cloud server accesses the file in the file system with the address $addr_{File}$ from the matching entry, and returns $C = \langle\{\mathcal{F}_1\}_{\mathrm{FEK}_1}, \{\mathrm{FEK}_1\}_\alpha, \ldots, \{\mathcal{F}_l\}_{\mathrm{FEK}_l}, \{\mathrm{FEK}_l\}_\alpha\rangle$ to the user. The user will recover the encrypted file with a decryption key α. The search index $\mathcal{SI}$ can be ordered according to a search key fid thus can be built into a tree-structure. The search operation cost with a search key fid then becomes $\mathrm{O}(\log n)$, where n denotes the size of $\mathcal{SI}$.

File Update: The procedure for updating a file is fully equal to *New File Upload* operation.

File Deletion: The data owner sends a file deletion request with fid. The cloud server searches a matching $\mathcal{SI}$ entry associated with fid. If the searched file is associated with more fids other than the requested one, the cloud server just removes the entry from $\mathcal{SI}$. Otherwise, the server also removes the actual file in the storage.

4.3 Efficiency Improvements

We present how to improve efficiency of the proposed scheme and to enhance its performance.

4.3.1 Auxiliary Search Index

The operations of *New File Upload* and *File Update* require a search operation on the search index $\mathcal{SI}$. Since $\mathcal{SI}$ entries cannot be sorted with respect to search keys CT and CT_n, the searching cost is linear to the size of $\mathcal{SI}$ (*i.e.*, the number of files). In this section, we describe a technique that improves the efficiency of the proposed scheme in terms of searching cost. An idea behind the technique is motivated from the fact that the size of uploading files is distributed between a few to giga bytes and the cloud server can get information of the size of a file from the encrypted one. Our idea is to use an auxiliary search index ($\mathcal{AI}$) in addition to $\mathcal{SI}$. In $\mathcal{AI}$, the size of a file is actually a search key, and an index value contains $addr_{Index}$, an address of the $\mathcal{SI}$ entry that indicates the corresponding file. Each $\mathcal{AI}$ entry can be sorted with respect to its search key. With help of a tree structure like B+-tree, an item insertion, deletion and searching operation in $\mathcal{AI}$ can be handled in an efficient and scalable manner.

When a data owner uploads (or updates) a file, the data owner sends a tuple $<size, fid, TK_{fid}, TK_{n,fid}, CT_{fid}, CT_{n,fid}>$ to the cloud server. Then, the server first looks up $\mathcal{AI}$ entries that correspond to *size*. If found, the cloud server gets matching $\mathcal{SI}$ entries which are addressed by $addr_{Index}$. The number of matching $\mathcal{SI}$ entries may be one or more according to a density of file size distribution. The cloud server then enumerates the matching $\mathcal{SI}$ entries and finds a corresponding entry such that $Test$ (or $Test_n$) algorithm results in 'Yes'. In this technique, using $\mathcal{AI}$ limits efficiently the search scope of $\mathcal{SI}$ thus eventually reduces searching cost to sub-linear complexity. In order to use this technique, the original scheme should be slightly modified. The only difference from the original scheme is that *size* field is added on the above tuple in this technique.

4.3.2 Tradeoff between Security and Network Efficiency

The proposed scheme is designed to be resistant against the online guessing attack. While an unauthorized user has difficulty in guessing the content of a stored file, usual file upload operation may also have little chance that a client-side deduplication event occurs, which eventually causes the increase of network transmission overhead. In order to avoid losing network transmission efficiency, the proposed scheme allows the security parameter d of SEPE$_n$ to be selected on the tradeoff between the security and the efficiency. The security parameter d denotes a domain size of permutation and actually determines the size of set of all permutations with a domain D such that $|D|=d$. Hence, the lower value of d gives higher probability that client-side deduplication occurs and vice versa.

This tradeoff is reasonable in practical applications. Data owners may upload not only high privacy requiring files that contain personal information but also common files like software setup files. The uploading files may also vary between high and low entropy. In the proposed scheme, the value of d is globally chosen in system setup phase. However, it is not difficult to modify the proposed scheme to allow d be selected for each file in uploading phase. Each file's security parameter d can be decided based on its prop-

erties like the entropy or the required level for privacy. File size can be another decision factor of d because it is reasonably assumed that file size and its entropy are highly correlated with each other [3]. For example, a file that requires no privacy or its size is greater than some threshold may be given its security parameter $d = 1$, which implies that client-side deduplication will always occur with probability 1 if the same file has existed in the cloud server.

A quantitative analysis of the relation between the security parameter d and both security and the network transmission overhead will be presented in Sect. 5.2.2.

4.4 Improving Reliability of the Cloud Storage

By applying a general data replication mechanism to the proposed scheme, we can further improve reliability of the scheme against malicious data owners incurring data loss. A data replication mechanism, where replication factor (RP) is r, creates r replicas of a data object and stores them across geographically distributed storage systems.

New File Upload operation of the proposed scheme is extended to utilize such a data replication as follows. When accepting $< fid, C >$ from a data owner, the cloud server searches on $\mathcal{SI}$ by using $Test$ algorithm. If a matching entry is found (*i.e.,* server-side deduplication event occurs), the cloud server gets $addr_{File}$ from the $\mathcal{SI}$ entry and determines the locations of existing replicas for the file. If the number of the existing replicas is less than r, a new one is allocated to store $C = \langle \{\mathcal{F}_1\}_{\mathrm{FEK}_1}, \{\mathrm{FEK}_1\}_\alpha, \ldots, \{\mathcal{F}_l\}_{\mathrm{FEK}_l}, \{\mathrm{FEK}_l\}_\alpha \rangle$. Otherwise, the cloud server selects one of them and replaces l_r blocks of the chosen replica, where l_r is randomly picked from $\{0, 1, \ldots, l\}$, by new blocks of C. In other cases, the cloud server follows the original protocol described in the previous section.

With help of an erasure (x, y)-code, corrupted data can be recovered as long as at least $\lceil lx/(x+y) \rceil$ blocks among r replicas stored within the storage sustain the corruption, where l is the number of blocks of the file.

Note that data replication is orthogonal to data deduplication. Goals of removing data redundancy and improving reliability can be achieved simultaneously using both data replication and data deduplication together.

5. Analysis

In this section, we first present the security analysis of the proposed scheme. Then, we present the storage, communication and computation performance analysis.

5.1 Security Analysis

5.1.1 Data Confidentiality

In the proposed scheme, files in the cloud storage are encrypted with a symmetric encryption algorithm like AES [26]. Encrypting the file generates a randomized ciphertext, because FEK is randomly chosen from the key space. Therefore any adversary including the cloud server who has only ciphertexts of the file cannot distinguish it from the random data. That is, the adversary does not know any information of the content of a file.

The cloud server will perform data deduplication through running $Test$ and $Test_n$ algorithm with inputs of tokens and ciphertexts of SEPE and SEPE$_n$. Unless tokens are given, the cloud server has negligible advantage in getting any information of the plaintext from its ciphertexts by theorems, **Theorems 1** and **2**. Although the cloud server may have knowledge of equivalence relations among the ciphertexts in the search index, **Theorem 3** says that any PPT algorithm including the test algorithms, $Test$ and $Test_n$, does not reveal any partial information to the cloud server other than the equivalence relation.

Note that the cloud server obviously can infer to the information of file size from the corresponding ciphertext because the file size and its ciphertext size are inherently correlated with each other.

Theorem 3. *For any two plaintexts M and M' such that $M \neq M'$, any PPT algorithm given SEPE (or SEPE$_n$) ciphertexts and tokens of M and M' as inputs cannot output any partial information of the plaintexts M and M' under the random oracle assumption.*

Proof. For the sake of simplicity, let us assume that hash functions used in SEPE and SEPE$_n$ (*i.e.,* H_1, H_2) are denoted as $F : \{0,1\}^* \rightarrow \mathbb{G}$. In computation of the tokens and the ciphertexts, the plaintext M and M' are given as inputs in evaluating the hash function F. By collision resistance property of a cryptographic hash function, it is highly unlikely that $F(M)=F(M')$ for any two different plaintext M and M'. Besides, using the fact that a cryptographic hash function can be modeled as a random oracle [27], both of $F(M)$ and $F(M')$ will be indistinguishable from random in $\mathbb{G}$. Therefore, in the case of $Test$ and $Test_n$ algorithm, an intermediate computation result γ will be indistinguishable from a random data in $\mathbb{G}_T$, and the test algorithm will eventually result in meaningless output 'No'. Another PPT algorithms given ciphertexts and tokens of M and M' as inputs will also eventually output a data that looks like a random from the view of the algorithm. □

5.1.2 Resistance against Online Guessing Attack

Now we analyze security against an adversary who mounts online guessing attack to figure out the information of interesting file in the cloud storage. The adversary is neither in possession of the file nor its hashed value. However, the adversary may have relatively small search domain for the file because the file has inherent low-entropy property or he has some knowledge of it. For each candidate in the search domain the adversary generates a token TK_n and then queries the token to the cloud server. If a matching ciphertext CT_n exists such that $Test_n(CT_n, TK_n)$ results 'Yes' for the query, the cloud server will respond with a message $FileExist$ in-

dicating deduplication (at client-side) event occurs. The adversary will continue querying to the cloud server until receiving $FileExist$.

In the attack scenario, an adversary may select a correct candidate in high likelihood of matching the target file. However, the adversary should query a correct token TK_n for the selected candidate in order to render a deduplication event (at client-side). **Theorem 4** says that a probability that an adversary computes a correct token TK_n such that $Test_n(CT_n, TK_n)$ results in 'Yes' is negligible in the security parameter d. Hence, the proposed scheme is resistant against online guessing attacks.

Theorem 4. *For any PPT algorithm $\mathcal{A}$, the probability that $\mathcal{A}$ outputs the corresponding token TK_n for given SEPE$_n$ ciphertext CT_n is negligible in the security parameter d.*

Proof. Let us assume a PPT algorithm $\mathcal{A}$ which computes a token TK_n for a corresponding SEPE$_n$ ciphertext CT_n. $\mathcal{A}$ is given d, g, α and CT_n as inputs and tries to find out the correct token TK_n. We also assume that CT_n $=\langle h^r, g^r, H_3(e(h, g^a)^r)\rangle$, where r is chosen at random from $\mathbb{Z}_p^*$. If $\mathcal{A}$ can get h from h^r, it is obvious that $\mathcal{A}$ easily computes the corresponding token $TK_n=\langle h^{k+\alpha}, g^k\rangle$ with given h, where k is chosen randomly from $\mathbb{Z}_p^*$. However, the problem that computes h from h^r in a cyclic group $\mathbb{G}$ is not easier than the discrete logarithm problem (DLP) on elliptic curves which is generally known to be computationally infeasible for any PPT algorithm if p is sufficiently large [23]. Hence, the best way for $\mathcal{A}$ to find out the token is to choose one of possible tokens randomly and test it by running $Test_n$ algorithm. $\mathcal{A}$ can choose a token at random by tossing coins to choose a random permutation $\pi : D \to D$, where $D = \{1, 2, \ldots, d\}$. Now we analyze the probability that $\mathcal{A}$ outputs the correct token TK_n given the input CT_n. Let us denote a set of all permutations $\pi : D \to D$ as Φ. We assume that CT_n has been computed with a permutation π. That is, $h = g^{\vec{u}\cdot\vec{v}_\pi}$ in CT_n, where $\vec{u} \cdot \vec{v}_\pi = \sum_{i\in D} iH_2^{\pi(i)}(M)$. When algorithm $\mathcal{A}$ chooses randomly a permutation π' from Φ, h is computed as $h = g^{\vec{u}\cdot\vec{v}_{\pi'}}$ in TK_n, where $\vec{u} \cdot \vec{v}_{\pi'} = \sum_{i\in D} iH_2^{\pi'(i)}(M)$. Note that $\vec{u} \cdot \vec{v}_\pi$ and $\vec{u} \cdot \vec{v}_{\pi'}$ also can be represented as $\vec{u} \cdot \vec{v}_\pi = pk + r$ and $\vec{u} \cdot \vec{v}_{\pi'} = pk' + r$, where p is order of $\mathbb{G}$, $k, k' \in \mathbb{N}$ and $0 \le r, r' < p$. Let us denote an event $p(k-1) \le \vec{u} \cdot \vec{v}_{\pi'} < pk$ as E_k. Then, the probability is

$$
\begin{aligned}
&\Pr\{\mathcal{A} \text{ outputs the correct } TK_n\} \\
&= \Pr\{\pi' \xleftarrow{R} \Phi\ ,\ Test_n(CT_n, TK_n) = \text{'Yes'}\ \} \\
&= \Pr\{\pi' \xleftarrow{R} \Phi\ ,\ g^{\vec{u}\cdot\vec{v}_\pi} = g^{\vec{u}\cdot\vec{v}_{\pi'}}\} \\
&= \Pr\{\pi' \xleftarrow{R} \Phi\ ,\ \vec{u} \cdot \vec{v}_\pi = \vec{u} \cdot \vec{v}_{\pi'} \bmod p\} \\
&\le \sum_{k\in\mathbb{N}} \Pr\{r = r'|E_k\}\Pr\{\pi' \xleftarrow{R} \Phi\ ,\ E_k\} \\
&\le \sum_{k\in\mathbb{N}} \Pr\{\pi' \xleftarrow{R} \Phi\ ,\ \pi = \pi'\}\Pr\{\pi' \xleftarrow{R} \Phi\ ,\ E_k\} \\
&= \Pr\{\pi' \xleftarrow{R} \Phi\ ,\ \pi = \pi'\} \sum_{k\in\mathbb{N}} \Pr\{\pi' \xleftarrow{R} \Phi\ ,\ E_k\} \\
&\le \Pr\{\pi' \xleftarrow{R} \Phi\ ,\ \pi = \pi'\} \\
&= \frac{1}{d!} \le \frac{1}{2^{d-1}}.
\end{aligned}
$$

Therefore, the probability that $\mathcal{A}$ outputs the corresponding TK_n is negligible in the security parameter d. □

5.1.3 Collusion-Resistance

Unauthorized users and even other data owners who do not have an access right to their interesting file can collude together to access the file. They should know the correct access token $\langle fid, \alpha\rangle$ in order to get the file via *File Access* operation. Hence they will try to compute $\langle fid, \alpha\rangle$ from information such as secret keys or tokens which they have. Assuming that fid is chosen from sufficiently large space and kept secret to the authorized users and data owners, the colluding adversaries cannot compute fid since fid is independent of any information which they have.

They may also try to compute the corresponding tokens TK, TK_n to take advantage of a side channel on client-side deduplication. However, since a permutation π of CT_n corresponding the file was chosen from Φ at random and is independent of any information, it is impossible for the colluding adversaries to get the file by **Theorem 4**.

5.1.4 Resistance against Malicious Data Owner

We analyze security of the proposed scheme against two kinds of attacks, which are launched by a malicious data owner who does not follow the protocol and behaves dishonestly.

Server-side deduplication disturbing attack: Let us consider a malicious data owner $\mathcal{M}$ who tries to disturb server-side deduplication so as to intentionally reduce the available storage resources of the cloud server. During *New File Upload* operation, $\mathcal{M}$ would mount such an attack by composing the wrong tuple $< fid, TK'_{fid}, TK_{n,fid}, CT'_{fid}, CT_{n,fid} >$, where TK'_{fid}, CT'_{fid} are calculated using the randomly generated secret key α' ($\neq \alpha$).

For the proposed scheme, this attack does not succeed, since the correctness of the received tuple is verified by the cloud server before acceptance. By definition of bilinear map (in Sect. 3.2.1) and definition of our construction, it is infeasible for $\mathcal{M}$ to compose the wrong tuple such that TK'_{fid}, CT'_{fid} are calculated using randomly generated α', while also satisfying the Eqs. (1)–(3) in Sect. 4.2, as well as allowing $Test(TK'_{fid}, CT'_{fid})$ to output 'Yes'.

Duplicate faking attack: We consider a malicious data owner $\mathcal{M}$ who tries to corrupt other users' data. During *New File Upload* operation, $\mathcal{M}$ may send a tuple $< fid, TK_{fid}, TK_{n,fid}, CT_{fid}, CT_{n,fid} >$ for $File$ correctly, but then upload a fake $< fid, C' >$, where C' is constructed from $File'(\neq File)$ or a wrong secret key $\alpha'(\neq \alpha)$. We analyze security against such an attack in terms of two properties; (i) corruption detectability and (ii) data recoverability.

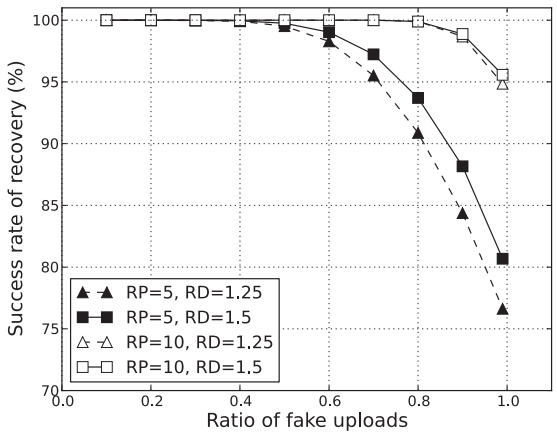


Fig. 3 File recoverability against fake uploads.

The proposed scheme ensures that the corruption of *File* is always detected by data owners or their authorized users who treat the same file. Let us denote a decryption of ciphertext E with a key k by $\mathsf{Dec}(k, E)$, n-th element of a tuple T by $T^{[n]}$, and a decoding function of an erasure code $\mathcal{E}$ by $\mathcal{D}$. Data corruption can be detected as follows.

1. get $C = \langle\{\mathcal{F}_1\}_{\mathrm{FEK}_1}, \{\mathrm{FEK}_1\}_\alpha, \ldots, \{\mathcal{F}_l\}_{\mathrm{FEK}_l}, \{\mathrm{FEK}_l\}_\alpha\rangle$ via *File Access* operation.
2. for each $T_i = \langle\{\mathcal{F}_i\}_{\mathrm{FEK}_i}, \{\mathrm{FEK}_i\}_\alpha\rangle$ in C, where $1 \le i \le l$, compute $v_i = \mathsf{Dec}(\mathsf{Dec}(\alpha, T_i^{[2]}), T_i^{[1]})$.
3. compute $w = \mathcal{D}(v_1 \| v_2 \| \ldots \| v_l)$, and $z = \mathit{KeyGen}(w)$.
4. test $z = \alpha$.

If z is not equal to a secret key α, then the file is corrupted.

With help of data replication, the proposed scheme also ensures that with high probability, a file can be recovered from corruption against fake uploads by $\mathcal{M}$. To justify our argument, we conducted a simulation, in which *New File Upload* operations for the same file (the block size is $b = 512$ and the number of blocks is $l = 16,384$) were performed in 1,000 times including fake uploads among them. The ratio of the fake uploads varies from 0 to 0.99. At each *New File Upload* operation, it was checked that sufficient blocks on the storage remain uncorrupted for recovery.

Figure 3 shows the simulation result. RP refers to a replication factor of data replication mechanism (*i.e.,* the number of replicas), and RD to a redundancy factor of the erasure code. With RP=10 and RD=1.5, the probability of successfully recovering corrupted data is more than 95%.

5.2 Performance Analysis

5.2.1 Storage Overhead

When a data owner uploads a file to the cloud server, data deduplication will always occur either at client-side or at server-side, if the same file exists in the cloud storage. That is, the proposed scheme always removes redundant copies of a file across multiple users, thus keeps the cloud storage optimized in terms of disk space utilization.

Additional storage overhead may be introduced due to the search index. The size of $\mathcal{SI}$ or $\mathcal{AI}$ entry, however, is negligible compared to the size of corresponding file.

Employing data replication, which is discussed in Sect. 4.4, may increase the required storage capacity. In the proposed scheme, however, data replication is run over deduplicated data, and thus is independent of eliminating the redundancy. The amount of increased storage resources does not exceed by the replication factor of a data replication mechanism.

5.2.2 Network Transmission Overhead

The hybrid approach that prevents the online guessing attack may introduce some network transmission overhead due to unnecessary file uploads. We analyze how much the network bandwidth is actually consumed for the proposed scheme.

For this, we compare the proposed scheme with a basic scheme, which is exactly same to the proposed scheme except that client-side deduplication always occurs (*i.e.,* the security parameter $d = 1$). Note that the basic scheme incurs no network transmission overhead at all. Thus, the difference Δ between the amount of network bandwidth consumed for the proposed scheme and the amount for the basic scheme represents the network transmission overhead of the proposed scheme.

Several events in *New File Upload* operation are defined as follows:

1. E: The file $\mathcal{F}$ exists in the cloud storage.
2. T_P: *New File Upload* operation transmits the actual copy of $\mathcal{F}$ in the proposed scheme.
3. T_B: *New File Upload* operation transmits the actual copy of $\mathcal{F}$ in the basic scheme.

From the security analysis in Sect. 5.1.2, we have $\Pr\{T_P \mid E\} \ge 1 - \frac{1}{d!}$. Without loss of generality, we assume $\Pr\{T_P \mid E\} = 1 - \frac{1}{d!}$ in this section. Note that $\Pr\{T_B \mid E\} = 0$. The probability that the event T_P occurs is

$$\begin{aligned}\Pr\{T_P\} &= \Pr\{T_P \mid E\}\Pr\{E\} + \Pr\{T_P \mid \neg E\}\Pr\{\neg E\} \\ &= \Pr\{T_P \mid E\}\Pr\{E\} + \Pr\{\neg E\} \\ &= \Pr\{T_P \mid E\}\Pr\{E\} + 1 - \Pr\{E\} \\ &= \left(1 - \frac{1}{d!} - 1\right)\Pr\{E\} + 1 = -\frac{\Pr\{E\}}{d!} + 1.\end{aligned}$$

By the similar way, the probability that the event T_B occurs is $\Pr\{T_B\} = 1 - \Pr\{E\}$.

The expected amount of network traffic in uploading the file $\mathcal{F}$ for the proposed scheme and the basic scheme are $s \cdot \Pr\{T_P\}$ and $s \cdot \Pr\{T_B\}$, respectively, where s is the size of $\mathcal{F}$. Thus, the network transmission overhead of the proposed scheme can be represented by the difference Δ

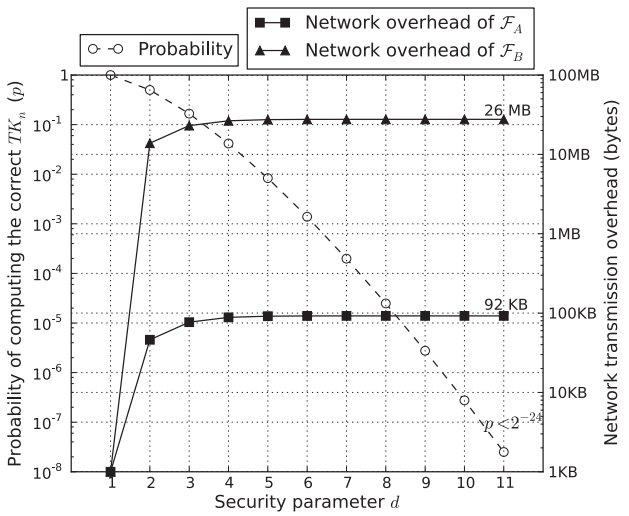


Fig. 4 Tradeoff between security and the network transmission overhead over the security parameter d (The file size of $\mathcal{F}_A$ and $\mathcal{F}_B$ is 90 MBs, $\Pr\{E_{\mathcal{F}_A}\} = 0.001$ and $\Pr\{E_{\mathcal{F}_B}\} = 0.3$).

$$\Delta = | s\Pr\{T_P\} - s\Pr\{T_B\} | = s\Pr\{E\}\left(1 - \frac{1}{d!}\right). \quad (4)$$

For each file in the cloud storage, the probability $\Pr\{E\}$ will be distributed between 0 to 1 according to its properties, especially to the type of file content. That is, it is easily inferred that common files (*e.g.,* software install files) usually have high $\Pr\{E\}$, and private files, that contain sensitive and personal information, requiring high privacy have low $\Pr\{E\}$. From the above equation, we can observe that as $\Pr\{E\}$ goes to 0, so does Δ, while Δ closes to $s(1 - 1/d!)$ as $\Pr\{E\}$ goes to 1. This result implies that we can reduce overall network transmission overhead substantially by just only decreasing the security parameter d of common files, while keeping private file's parameter d high, as we described in Sect. 4.3.2.

Let us consider an example that a data owner uploads two files $\mathcal{F}_A$ and $\mathcal{F}_B$, which represent a private file and a common file, respectively. Suppose that the size of both $\mathcal{F}_A$ and $\mathcal{F}_B$ is 90 MBs and the probabilities that the file exists in the cloud storage are $\Pr\{E_{\mathcal{F}_A}\} = 0.001$ and $\Pr\{E_{\mathcal{F}_B}\} = 0.3$. Figure 4 shows the quantitative relation between the security parameter d and both security (*i.e.*, probability $p = \frac{1}{d!}$ that an adversary computes the correct TK_n) and the network transmission overhead ($\Delta_{\mathcal{F}_A}$ for $\mathcal{F}_A$ and $\Delta_{\mathcal{F}_B}$ for $\mathcal{F}_B$). Setting the parameter d to 11 for both $\mathcal{F}_A$ and $\mathcal{F}_B$ raises the overall network transmission overhead ($\Delta_{\mathcal{F}_A} + \Delta_{\mathcal{F}_B}$) to more than 26 MBs. On the other hand, the overall overhead can be reduced to 92 KBs by just only decreasing the common file $\mathcal{F}_B$'s parameter d to 1, while keeping the probability p of the private file $\mathcal{F}_A$ lower than 2^{-24}.

5.2.3 Computation Overhead

Computational burden at the cloud server is mainly introduced by searching over the storage. We analyze the computation complexity for the following operations that perform the searching: *New File Upload*, *File Update*, *File Access*.

Table 3 Computation complexity of the proposed scheme considering two strategies: $\mathcal{SI}$ and $\mathcal{SI}$ with $\mathcal{AI}$ ($\mathcal{SI}$+$\mathcal{AI}$).

Operation	The Proposed Scheme	
	$\mathcal{SI}$	$\mathcal{SI}$+$\mathcal{AI}$
New File Upload	$O(N_s)$	$O(\log N_a + N_{s,a})$
File Update	$O(N_s)$	$O(\log N_a + N_{s,a})$
File Access	$O(\log N_s)$	$O(\log N_s)$

In *New File Upload* and *File Update* operation, the cloud server searches the matching file in $\mathcal{SI}$ through two search keys CT and CT_n. For each $\mathcal{SI}$ entry the cloud server compares it with given tokens by computing $Test$ and $Test_n$ algorithms which require two pairings and one multiplication over $\mathbb{G}_T$. Let us denote N_s as the total number of $\mathcal{SI}$ entries, N_a as the total number of $\mathcal{AI}$ entries and $N_{s,a}$ as the number of $\mathcal{SI}$ entries which correspond to the matching $\mathcal{AI}$ entry. The computation complexity of *New File Upload* and *File Update* are shown in Table 3. With help of $\mathcal{AI}$, the proposed scheme ($\mathcal{SI}$+$\mathcal{AI}$ in Table 3) can have sub-linear complexity in this operations.

In *File Access* operation, $\mathcal{SI}$ is retrieved through the search key fid. Since $\mathcal{SI}$ is sorted according to fid, the searching cost is more efficient than that of above operations as shown in Table 3. We also note that no cryptographic operations are needed in comparison over fid.

6. Related Work

Existing approaches related to our proposed scheme include 1) secure data outsourcing and 2) data deduplication using convergent encryption.

Most of them addressed the issue of secure data outsourcing in terms of cryptographic access control as well as searching over encrypted data. In order to achieve a goal of fine-grained access control and efficient revocation over the outsourced data, various techniques are proposed built on attribute-based encryption (ABE) [28], [29], which is a cryptographic primitive that ensures access control over encrypted data. S. Yu *et al.* addressed an issue of attribute revocation and proposed two solutions [5] and [4] in CP (Ciphertext Policy)-ABE and KP (Key Policy)-ABE, respectively. J. Hur *et al.* [6] combines CP-ABE and broadcast encryption technique [25] to enable more fine-grained access control with efficient and scalable revocation capability. Z. Zhou *et al.* [7] and M. Green *et al.* [8] considered ABE encryption and decryption overhead at client side, which grows with the complexity of the access formula. They proposed solutions that outsource the computation burden to the cloud server while preserving privacy of outsourced data.

Besides an issue of access control, it is also important to solve the problem of searching over encrypted data in secure data outsourcing. D. Boneh *et al.* [9] first described the notion of predicate encryption and gave PEKS, a concrete implementation of equality predicate encryption in the public-key setting. Their solution is suitable for a secure e-mail server which is possible for the untrusted server to re-

trieve encrypted mails. Several approaches addressed issues of efficient searching over encrypted data through tree-based indexing. M. Bellare *et al.* [11] and S. Sedghi *et al.* [12] proposed deterministic encryption in which the searching cost is $O(\log n)$ in the public-key setting and the symmetric-key setting, respectively. Due to a deterministic property, these solutions [11], [12] loose some security, as noted in [10]. R. Curtmola *et al.* [10] and C. Dong *et al.* [13] focused on enhancing security notions of previous solutions.

The notion of convergent encryption was first presented in [16] as a rough idea. Convergent encryption uses a hashed value computed from a file as an encryption key of the file itself. Any client encrypting a given data chunk will use the same key to do so, thus identical plaintext values will encrypt to identical ciphertext values, regardless of who encrypts them. This helps the storage server perform data deduplication on the encrypted data. Several solutions after [16] fundamentally follow this strategy. M. Storer *et al.* [15] further extended [16] and implemented data deduplication framework on the encrypted data storage. L. Marques *et al.* [17] and P. Anderson *et al.* [18] utilized convergent encryption to address different applications such as mobile devices. The security of convergent encryption cannot be guaranteed strongly due to its inherent deterministic property, and even has not been rigorously analyzed yet in the previous approaches.

7. Conclusion

This paper aims at achieving both cost efficiency and data security in cloud computing. One challenge in this context is to build a data deduplication system which offers cost savings in terms of disk space and network bandwidth utilization, while also providing data security and privacy against the untrusted cloud server and the unauthorized users. In this paper, we proposed an efficient and secure data deduplication scheme to resolve the challenging issue. In order to achieve both of efficiency and security, we constructed two equality predicate encryption schemes in the symmetric-key setting, on which the proposed data deduplication scheme is built. Our rigorous security proofs show that our proposed scheme is provably secure under cryptographic security models.

Acknowledgements

This research was funded by the MSIP (Ministry of Science, ICT & Future Planning), Korea in the ICT R&D Program 2013.

References

[1] M. Armbrust, A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, et al., "Above the clouds: A berkeley view of cloud computing," Technical Report UCB/EECS-2009-28, 2009.

[2] D. Russell, "Data deduplication will be even bigger in 2010," Gartner, Feb. 2010.

[3] D. Harnik, B. Pinkas, and A. Shulman-Peleg, "Side channels in cloud services: Deduplication in cloud storage," IEEE Security and Privacy Magazine, vol.8, pp.40–47, Nov. 2010.

[4] S. Y, C. Wang, K. Ren, and W. Lou, "Achieving secure, scalable, and fine-grained data access control in cloud computing," Proc. IEEE Conf. Information Comm. (INFOCOM'10), pp.534–542, 2010.

[5] S. Yu, C. Wang, K. Ren, and W. Lou, "Attribute based data sharing with attribute revocation," Proc. ACM Symp. Information, Computer and Comm. Security (ASIACCS'10), pp.261–270, 2010.

[6] J. Hur and D.K. Noh, "Attribute-based access control with efficient revocation in data outsourcing systems," IEEE Trans. Parallel Distrib. Syst., vol.22, pp.1214–1221, July 2011.

[7] Z. Zhou and D. Huang, "Efficient and secure data storage operations for mobile cloud computing." Cryptology ePrint Archive, Report 2011/185, 2011. http://eprint.iacr.org/.

[8] M. Green, S. Hohenberger, and B. Waters, "Outsourcing the decryption of abe ciphertexts," Proc. USENIX Conf. Security (SEC'11), 2011.

[9] D. Boneh, G. Di Crescenzo, R. Ostrovsky, and G. Persiano, "Public key encryption with keyword search," Proc. Eurocrypt'04, pp.506–522, 2004.

[10] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky, "Searchable symmetric encryption: improved definitions and efficient constructions," Proc. ACM Conf. Computer and Comm. Security (CCS'06), pp.79–88, 2006.

[11] M. Bellare, A. Boldyreva, and A. O'Neill, "Deterministic and efficiently searchable encryption," Proc. CRYPTO'07, pp.535–552, 2007.

[12] S. Sedghi, P. van Liesdonk, J.M. Doumen, P.H. Hartel, and W. Jonker, "Adaptively secure computationally efficient searchable symmetric encryption," Technical Report TR-CTIT-09-13, Centre for Telematics and Information Technology University of Twente, April 2009.

[13] C. Dong, G. Russello, and N. Dulay, "Shared and searchable encrypted data for untrusted servers," Journal of Comput. Secur., vol.19, pp.367–397, Aug. 2011.

[14] M. Mulazzani, S. Schrittwieser, M. Leithner, M. Huber, and E. Weippl, "Dark clouds on the horizon: using cloud storage as attack vector and online slack space," Proc. USENIX Conf. Security (SEC'11), 2011.

[15] M.W. Storer, K. Greenan, D.D. Long, and E.L. Miller, "Secure data deduplication," Proc. ACM Int'l Workshop on Storage security and survivability (StorageSS'08), pp.1–10, 2008.

[16] 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. Int'l Conf. Distributed Computing Systems (ICDCS '02), 2002.

[17] L. Marques and C.J. Costa, "Secure deduplication on mobile devices," Proc. Workshop on Open Source and Design of Communication (OSDOS'11), pp.19–26, 2011.

[18] P. Anderson and L. Zhang, "Fast and secure laptop backups with encrypted de-duplication," Proc. Int'l Conf. Large installation system administration (LISA'10), pp.1–8, 2010.

[19] S. Goldwasser and S. Micali, "Probabilistic encryption," Journal of computer and system sciences, vol.28, no.2, pp.270–299, 1984.

[20] D. Boneh, A. Sahai, and B. Waters, "Functional encryption: definitions and challenges," Proc. Conf. Theory of cryptography (TCC'11), pp.253–273, 2011.

[21] E. Shen, E. Shi, and B. Waters, "Predicate privacy in encryption systems," Proc. Conf. Theory of Cryptography (TCC'09), pp.457–473, 2009.

[22] D. Boneh and M.K. Franklin, "Identity-based encryption from the weil pairing," Proc. CRYPTO'01, pp.213–229, 2001.

[23] A. Joux, "The weil and tate pairings as building blocks for public key cryptosystems," Proc. Int'l Symp. Algorithmic Number Theory (ANTS'05), pp.20–32, 2002.

[24] C. Blundo, V. Iovino, and G. Persiano, "Private-key hidden vector

encryption with key confidentiality," Proc. Cryptography and Network Security 2009 (CANS'09), pp.259–277, 2009.
[25] D. Naor, M. Naor, and J.B. Lotspiech, "Revocation and tracing schemes for stateless receivers," Proc. CRYPTO'01, pp.41–62, 2001.
[26] J. Daemen and V. Rijmen, The design of Rijndael: AES-the advanced encryption standard, 2002.
[27] M. Bellare and P. Rogaway, "Random oracles are practical: a paradigm for designing efficient protocols," Proc. ACM Conf. Computer and Comm. Security (CCS'93), pp.62–73, 1993.
[28] V. Goyal, O. Pandey, A. Sahai, and B. Waters, "Attribute-based encryption for fine-grained access control of encrypted data," Proc. ACM Conf. Computer and Comm. Security (CCS'06), pp.89–98, 2006.
[29] J. Bethencourt, A. Sahai, and B. Waters, "Ciphertext-policy attribute-based encryption," Proc. IEEE Symp. Security and Privacy (SP'07), pp.321–334, 2007.

Appendix A: Proof of Theorem 1

Proof. Suppose $\mathcal{A}$ is an adversary algorithm that tries to break SEPE. $\mathcal{A}$ may run in two ways; (1) guessing a binary string *aux* to extract the secret key and (2) performing the security game described in Sect. 3.1.2.

Let us denote by $p(\cdot)$ a polynomial function. Since the min-entropy of the distribution of *aux* is $H_\infty^{aux} = p(\lambda)$, $\mathcal{A}$'s probability of guessing the correct *aux* is $2^{-p(\lambda)}$. Thus, extracting the correct secrete key is infeasible.

The only way to break SEPE is by the security game. Suppose that $\mathcal{A}$ has advantage ϵ in breaking SEPE by running the security game. We show that an algorithm $\mathcal{B}$ that solves the BDH problem can be constructed using algorithm $\mathcal{A}$. Suppose $\mathcal{A}$ makes at most q_C ciphertext queries, q_T tokens queries and at most q_{H_3} hash function queries to H_3. Then, the advantage of algorithm $\mathcal{B}$ is at least $\epsilon' = \epsilon/eq_{H_3}(q_T + q_C)$, where e is the base of the natural logarithm. If the BDH assumption holds in $\mathbb{G}$, then ϵ' is negligible in λ and consequently ϵ must be negligible in λ. Let g be a generator of $\mathbb{G}$. Algorithm $\mathcal{B}$ is given $g, u_1 = g^\alpha, u_2 = g^\beta, u_3 = g^\gamma \in \mathbb{G}$. $\mathcal{B}$ simulates the challenger and interacts with algorithm $\mathcal{A}$ to output $v = e(g,g)^{\alpha\beta\gamma} \in \mathbb{G}_T$ as follows:

Setup: Algorithm $\mathcal{B}$ gives $\mathcal{A}$ the global parameter: $\langle p, g, \mathbb{G}, \mathbb{G}_T \rangle$.

H_1-queries: Algorithm $\mathcal{A}$ can make queries to the random oracle H_1 at any time. $\mathcal{B}$ maintains the H_1-list which is a list of tuples $\langle M_i, h_i, a_i, k_i, r_i, c_i \rangle$ to respond to the query. The list is initially empty. Here i represents the sequence of the queries ($1 \le i \le q_C + q_T$). For each query $M_i \in \{0,1\}^*$, algorithm $\mathcal{B}$ responds as follows:

1. If the query M_i is already in the H_1-list in a tuple $\langle M_i, h_i, a_i, k_i, r_i, c_i \rangle$ then $\mathcal{B}$ responds with $H_1(M_i)$=$h_i \in \mathbb{G}$.
2. Otherwise, $\mathcal{B}$ picks a random $c_i \in \{0,1\}$ so that $\Pr\{c_i = 0\}$=$1/(q_C + q_T + 1)$, and also picks $a_i, k_i, r_i \in \mathbb{Z}_p^*$ at random.
   If $c_i = 0$, then $\mathcal{B}$ computes $h_i = u_2 g^{a_i}$.
   If $c_i = 1$, then $\mathcal{B}$ computes $h_i = g^{a_i}$.
3. Algorithm $\mathcal{B}$ appends the tuple $\langle M_i, h_i, a_i, k_i, r_i, c_i \rangle$ to the H_1-list and sends $H_1(M_i)$=h_i to $\mathcal{A}$.

H_3-queries: To respond to these queries, $\mathcal{B}$ maintains a list of tuples $\langle t_i, V_i \rangle$ called the H_3-list. The list is initially empty. If the query t_i already appears on the H_3-list then $\mathcal{B}$ responds with $H_3(t_i)$=V_i. Otherwise, $\mathcal{B}$ responds with $H_3(t_i) = V$ by generating $V \in \mathbb{G}_T$ at random and adds the new tuple $\langle t_i, V \rangle$ to the H_3-list.

Ciphertext queries: When $\mathcal{A}$ issues a query for the ciphertext of M_i, $\mathcal{B}$ responds as follows:

1. $\mathcal{B}$ responds to H_1-queries by running the above algorithm to obtain h_i=$H_1(M_i)$. If c_i=0 then $\mathcal{B}$ outputs $\perp$ and halts. If c_i=1 then $h_i = g^{a_i} \in \mathbb{G}$. Construct $h_i^{r_i}, g_i^{r_i}$ and E=$\mathrm{e}(h_i, u_1)^{r_i}$.
2. $\mathcal{B}$ also gets $H_3(E)$ for the query E by running the above algorithm and gives the correct ciphertext CT=$\langle h_i^{r_i}, g_i^{r_i}, H_3(E) \rangle$ to $\mathcal{A}$.

Token queries: When $\mathcal{A}$ issues a query for the token corresponding to M_i, $\mathcal{B}$ responds as follows:

1. Similar to the ciphertext query, $\mathcal{B}$ gets h_i=$H_1(M_i)$ for M_i by running the above algorithm.
2. If c_i=0 then $\mathcal{B}$ outputs $\perp$ and halts. If c_i=1 then $h_i = g^{a_i} \in \mathbb{G}$. $\mathcal{B}$ gives the token $TK = \langle h_i^{k_i} u_1^{a_i}, g^{k_i} \rangle$ to algorithm $\mathcal{A}$. Observe that TK is correct because $h_i^{k_i} u_1^{a_i} = h_i^{k_i} h_i^{\alpha} = h_i^{k_i+\alpha}$.

Challenge: Algorithm $\mathcal{A}$ produces a pair of challenging plaintext M_0 and M_1. $\mathcal{B}$ generates the challenge as follows:

1. Algorithm $\mathcal{B}$ runs the above algorithm for responding to H_1-queries to obtain a $h_0, h_1 \in \mathbb{G}$ such that $H_1(M_0)$=h_0 and $H_1(M_1)$=h_1. If both $c_0 = 1$ for h_0 and $c_1 = 1$ for h_1 then $\mathcal{B}$ outputs $\perp$ and halts.
2. Otherwise, at least one of c_0, c_1 is equal to 0. Algorithm $\mathcal{B}$ picks a $b \in \{0,1\}$ at random such that $c_b = 0$.
3. Algorithm $\mathcal{B}$ responds with the challenge ciphertext CT=$\langle I, u_3, J \rangle$ for a random $I \in \mathbb{G}$ and $J \in \mathbb{G}_T$. Observe that the challenge implicitly defines I, J as follows:

$$\begin{aligned} I &= H_1(M_b)^\gamma, \\ J &= H_3(e(H_1(M_b), u_1^\gamma)) = H_3(e(u_2 g^{a_b}, g^{\alpha\gamma})) \\ &= H_3(e(g,g)^{\alpha\gamma(\beta+a_b)}). \end{aligned}$$

Hence, the challenge is a valid ciphertext for M_b.

More ciphertexts, token queries: $\mathcal{A}$ continues to ask for the ciphertext or the token of M_i of his choice except M_0, M_1. Algorithm $\mathcal{B}$ responds to these queries as before.

Output: Finally, $\mathcal{A}$ outputs its guess $b \in \{0,1\}$. Then, $\mathcal{B}$ picks randomly a tuple $\langle t, V \rangle$ from the H_3-list and outputs $t/e(u_1, u_3)^{a_b}$ as its guess for $e(g,g)^{\alpha\beta\gamma}$.

This completes the description of algorithm $\mathcal{B}$. Now we analyze the probability that $\mathcal{B}$ does not output $\perp$ during the simulation. We define two events; E_1 denotes an event that $\mathcal{B}$ does not output $\perp$ during token or ciphertext queries, and E_2 denotes an event that $\mathcal{B}$ does not output $\perp$ during the challenge phase. The probability that a ciphertext or a token

query cause $\mathcal{B}$ to output $\perp$ is $\Pr\{c_i = 0\} = 1/(q_C + q_T + 1)$. Since $\mathcal{A}$ makes at most $q_C + q_T$ of all queries the probability that event E_1 occurs is at least $(1 - 1/(q_C + q_T + 1))^{(q_C+q_T)} \geq 1/e$. That is, $\Pr\{E_1\} \geq 1/e$. Also, $\mathcal{B}$ will output $\perp$ in the challenge phase if $\mathcal{A}$ produces M_0, M_1 such that $c_0 = c_1 = 1$. Since $\Pr\{c_i = 0\} = 1/(q_C + q_T + 1)$ for $i = 0, 1$, and the two values are independent, $\Pr\{c_0 = c_1 = 1\} = (1 - 1/(q_C + q_T + 1))^2 \leq 1 - 1/(q_C + q_T)$. Therefore, $\Pr\{E_2\} = 1 - \Pr\{c_0 = c_1 = 1\} \geq 1/(q_C + q_T)$. These two events E_1 and E_2 are independent because $\mathcal{A}$ can never issue a ciphertext or token query for the challenge plaintext, M_0 and M_1. Therefore, the probability that $\mathcal{B}$ does not output $\perp$ during the simulation is $\Pr\{E_1 \wedge E_2\} = \Pr\{E_1\}\Pr\{E_2\} \geq 1/e(q_C + q_T)$.

Next, assuming that $\mathcal{B}$ does not output $\perp$, we argue that during the simulation $\mathcal{A}$ issues a query for $H_3(e(H_1(M_b), u_1^{\gamma}))$ with probability at least ϵ as shown in the proof of Theorem 3.1 of [9], where M_b is the challenge plaintext. That is, the value $H_3(e(H_1(M_b), u_1^{\gamma})) = e(g^{\beta+a_b}, g)^{\alpha\gamma}$ will appear on some tuple in the H_3-list with probability at least ϵ. Algorithm $\mathcal{B}$ will choose the correct tuple with probability at least $1/q_{H_3}$ and therefore, it will produce the correct answer with probability at least ϵ/q_{H_3} assuming that algorithm $\mathcal{B}$ does not output $\perp$.

Consequently, since $\mathcal{B}$ does not output $\perp$ with probability at least $1/e(q_C + q_T)$ the probability that algorithm $\mathcal{B}$ succeeds overall is at least $\epsilon' = \epsilon/eq_{H_3}(q_C + q_T)$. □

Appendix B: Proof of Theorem 2

Proof. The difference between SEPE and SEPE$_n$ is that in SEPE$_n$ scheme security parameter d, a random permutation π and hash functions H_2^i are additionally defined. Without loss of generality, we can assume that the security parameter d is a constant and a random permutation π is fixed. Then the difference between SEPE and SEPE$_n$ lies only in the computation of h. In the SEPE$_n$ scheme, h is calculated as $h = g^s$, where $s = \vec{u} \cdot \vec{v}_\pi = \sum_{i \in D} iH_2^{\pi(i)}(M)$, and $H_2^i : \{0,1\}^* \rightarrow \mathbb{Z}_p^*$ are d hash functions, while h is computed as h=$H_1(M)$ in the SEPE scheme, where H_1 is a hash function $H_1 : \{0,1\}^* \rightarrow \mathbb{G}$. In the random oracle model, all hash functions are treated as the random oracle and the distribution of hashed values is uniform. In the SEPE$_n$ scheme, evaluations of the random oracles H_2^i are independent with each other. Hence, the distribution of s is uniform in $\mathbb{N}$, which implies that the distribution of $h = g^s$ is also uniform in $\mathbb{G}$. From the view of an adversary, these two schemes are statistically indistinguishable because in the SEPE scheme the value of hash function H_1 is also distributed uniformly in $\mathbb{G}$. Therefore, assuming SEPE scheme is semantically secure against an adaptively chosen plaintext attack, SEPE$_n$ scheme is also semantically secure against the attack. And, by **Theorem 1**, SEPE$_n$ scheme is secure. □

Youngjoo Shin received the B.S. degree in Computer Science and Engineering from Korea University, Korea in 2006 and the M.S. degree in Computer Science from Korea Advanced Institute of Science and Technology, Korea in 2008. He is currently a Ph.D. candidate at Computer Science Department in Korea Advanced Institute of Science and Technology, Korea, and is also a researcher at The Attached Institute of ETRI, Korea. His research interests include cryptography, network security and cloud computing security.

Kwangjo Kim received the B.S. and M.S. degrees of Electronic Engineering in Yonsei University, Korea, and Ph.D. of Div. of Electrical and Computer Engineering in Yokohama National University, Japan. Currently he is a professor at Computer Science Department in Korea Advanced Institute of Science and Technology, Korea. He served the president of Korean Institute on Information Security and Cryptography (KIISC) in 2009 and Board Member of IACR from 2000 to 2004. His research interests include the theory of cryptography and information security and their practice.