2015 10th Asia Joint Conference on Information Security

Encrypted Data Deduplication in Cloud Storage

Chun-I Fan
Department of Computer Science and Engineering, National Sun Yat-sen University, Kaohsiung, Taiwan
Email: cifan@faculty.nsysu.edu.tw

Shi-Yuan Huang
CyberTrust Technology Institute
Institute for Information Industry
Taipei, Taiwan
Email: shiyuan.huang@gmail.com

Wen-Che Hsu
Department of Computer Science and Engineering, National Sun Yat-sen University, Kaohsiung, Taiwan
Email: a0072362170@gmail.com

***Abstract*—Cloud storage is a remote storage service, where users can upload and download their data anytime and anywhere. However, it raises issues regarding privacy and data confidentiality because all the data are stored in the cloud storage. This is a subject of concern for users, and it affects their willingness to use cloud storage services. On the other hand, a cloud storage server typically performs a specialized data compression technique (data deduplication) to eliminate duplicate data because the storage space is not infinite. Data deduplication, which makes it possible for data owners to share a copy of the same data, can be performed to reduce the consumption of storage space. Due to the above issues, there is a research on encrypted data deduplication. In this manuscript, we propose an encrypted data deduplication mechanism which makes the cloud storage server be able to eliminate duplicate ciphertexts and improves the privacy protection.**

I. Introduction

In recent years, cloud computing has become a hot topic and brings many advantages through various services. The complex hardware, database, and operating system can be handled by a cloud server. Users only need some simple devices, which can connect to the cloud server. However, in the environment, the cloud server can obtain and control all the uploaded data because all the data are stored or operated in the cloud. The security and privacy issues are very important in cloud computing [1], [2]. In order to protect privacy, users encrypt their data by some encryption algorithms and upload the encrypted data to the cloud. As a result, extensive research has been performed in cloud computing. (e.g., data integrity check, searchable encryption, access control, and private operation)

We focus on the cloud storage service. Users can store their data in the cloud storage and download the stored data anywhere. Even if users exhaust their own storage spaces, the cloud storage server can expand the storage spaces without destroying the stored data. However, the fast growth of storage requirements burdens the cloud storage, which is not infinite. The cloud storage server typically applies the data deduplication technique [3], [4] to reduce the consumption of storage space.

At the same time, the goal of encryptions is to keep information secret and make it difficult to distinguish the encrypted data (i.e., ciphertexts) from random values. If an encryption is secure, it would be hard to obtain information from ciphertexts. Hence, encrypted data deduplication becomes a challenge because the first step of data deduplication is to search for duplicate data. If the cloud storage server can identify whether the contents of two ciphertexts are the same, it means that the ciphertexts leak some information about the plaintexts.

A. Overview of Data Deduplication

Data deduplication is a specialized data compression technique which makes all the data owners, who upload the same data, share a single copy of duplicate data and eliminates the duplicate copies in the storage. When users upload their data, the cloud storage server will check whether the uploaded data have been stored or not. If the data have not been stored, it will be actually written in the storage; otherwise, the cloud storage server only stores a pointer, which points to the first stored copy, instead of storing the whole data. Hence, it can avoid the same data being stored repeatedly.

Generally, data deduplication can be divided into two basic approaches [5], [6]: the target-based data deduplication and the source-based data deduplication. The two approaches of data deduplication are described as follows.

- **Target-Based Data Deduplication:** In this approach, users simply upload their data, and the steps of data deduplication (searching and eliminating duplicate data) are handled by the cloud storage server. The target-based approach can improve the storage utilization, and users do not have to change their habits of using cloud storage services. However, the target-based approach only focuses on avoiding storing duplicate data. Those duplicate data are still uploaded repeatedly. Therefore, it cannot improve the volume of transmissions.
- **Source-Based Data Deduplication:** In this approach, users have to upload the identification of their data and query the cloud storage server whether the data are stored in the cloud storage before really uploading them. If the data have not been stored, users need to upload the whole data, and the cloud storage server completely stores them. Otherwise, users need to upload only the metadata, and the cloud storage server simply creates a pointer, which points to the first stored copy. Therefore, the source-based approach can improve both the utilization of the storage and the bandwidth. Nevertheless, it changes the familiar process of cloud storage services. When users want to upload their data, they must query the cloud storage server for the existence of the data first.

978-1-4799-1989-5/15 $31.00 © 2015 IEEE
DOI 10.1109/AsiaJCIS.2015.12

B. Security in Data Deduplication

Since all the data are uploaded to the cloud storage, users attend to the privacy issue in cloud storage service. We analyze two types of attackers in data deduplication as follows:

- **Malicious User:** It is a special attacker in source-based data deduplication. In a source-based data deduplication scheme, each user queries the cloud storage server whether the data have been uploaded by another. The cloud storage server responds "Yes" or "No" honestly to avoid the duplicate data being uploaded repeatedly. Therefore, a malicious user can use the response of the cloud storage server to obtain private information about the existence of data.
- **Cloud Storage Server:** In cloud storage service, the cloud storage server can obtain and control all the uploaded data. In addition, if encrypted data deduplication can be performed, even though all the data are encrypted, the cloud storage server can still analyze the encrypted data by using the encrypted data deduplication and try to obtain information about the plaintexts.

II. Related Works

Many data deduplication mechanisms have been proposed recently [5], [6], [7], [8], [9], [10], [11], [12], [13], [14]. In this section, we research into the encrypted data deduplication and discuss some security issues.

A. Data Deduplication on Ciphertext

The first step of data deduplication is to search duplicate data in a cloud storage, and the cloud storage server only saves one copy of the duplicate data, which can be shared by the data owners. This process is easy when all the stored data are unencrypted. However, when the data owners encrypt their data using their secret keys for privacy consideration, the above-mentioned data deduplication becomes infeasible. There are two ways that can launch data deduplication on encrypted data:

- Using data deduplication on ciphertexts is infeasible because ciphertexts are distinct even though the corresponding plaintexts are the same. Therefore, the first way is to add a header. The header can let the cloud storage server be able to identify whether two different ciphertexts correspond to the same plaintext. However, the duplicate ciphertexts, which correspond to the same plaintext, were generated by different data owners using different secret keys. There is another problem: how can the cloud storage server save a shared ciphertext such that all the data owners can decrypt it, in which the cloud storage server cannot know the corresponding plaintext?
- The second way is to let the same plaintexts correspond to the same ciphertexts. In order to avoid the cloud storage server obtaining the plaintexts, the secret keys must be generated from the plaintexts, and thus the input of the encryption algorithm has no random factor. It means that the same plaintexts generate the same secret keys, and using the same secret keys to encrypt the same plaintexts will produce the same ciphertexts. Therefore, the cloud storage server can identify duplicate ciphertexts and just saves one copy of them. On the other hand, the data owners can decrypt the stored copy because duplicate ciphertexts correspond to the same secret key.

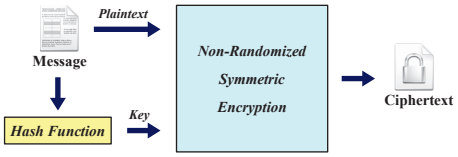

Fig. 1. Convergent Encryption

B. Convergent Encryption

Convergent encryption was first mentioned by John Pettitt on the cypherpunk's mailing list in 1996 [15]. It uses the hash values of plaintexts to be the secret keys of a non-randomized symmetric encryption. The same plaintexts generate the same secret keys and produce the same ciphertexts. Therefore, convergent encryption is generally used in data deduplication [7], [8], [13], [14], where the cloud storage server can search duplicate ciphertexts directly.

However, convergent encryption has been already known to have weaknesses on data confidentiality. The ciphertexts leak out some information about the plaintexts because the duplicate ciphertexts correspond to the same plaintexts.

In short, convergent encryption cannot completely preserve the data confidentiality. Using convergent encryption in the data deduplication can solve the problem of identifying duplicate ciphertexts, but cannot prevent data leakage.

III. Encrypted Data Deduplication Mechanism

In this section, we propose an encrypted data deduplication mechanism (EDDM), and it can be divided into two steps: $Cipher\ Structure\ Construction$, which protects the data confidentiality of the uploaded data; $Encrypted\ Data\ Deduplication$, which makes the cloud storage server be able to eliminate duplicate cipher structures.

A. Preliminaries

There are two encryption algorithms in EDDM. One of them is a symmetric encryption, which is used to encrypt the uploaded files; we take the AES encryption to be the symmetric encryption in the proposed mechanism. The other one is a multiplicative asymmetric homomorphic encryption, which is used to encrypt AES keys. In addition, there are some notations as follows:

- HL: a hash list, which is kept by the cloud storage server. It contains the hash values and the pointers of all stored cipher structures.
- K_j: a one-time AES key, which is randomly selected by a user and used to encrypt the uploaded file.
- (PK_{U_i}, SK_{U_i}): the public-private key pair of user U_i (a multiplicative asymmetric homomorphic encryption).

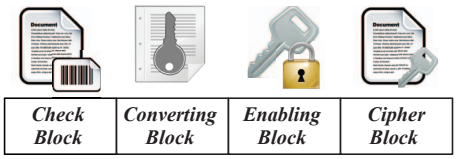

Fig. 2. Cipher Structure of EDDM

- $Hash_1(.), Hash_2(.)$: two cryptographic hash functions, $Hash_1$ and $Hash_2$: $\{0,1\}^* \rightarrow \{0,1\}^n$, where n is a pre-defined positive integer.
- $AES_{Enc}(K_j, m)$: the encryption algorithm of the AES encryption, which encrypts a message m with a symmetric key K_j.
- $AES_{Dec}(K_j, c)$: the decryption algorithm of the AES encryption, which decrypts a ciphertext c with a symmetric key K_j.
- $HE_{Enc}(PK_{U_i}, m)$: the encryption algorithm of a multiplicative asymmetric homomorphic encryption, which encrypts a message m with a public key PK_{U_i}.
- $HE_{Dec}(SK_{U_i}, c)$: the decryption algorithm of a multiplicative asymmetric homomorphic encryption, which decrypts a ciphertext c with a private key SK_{U_i}.
- $\odot$: the operator of a multiplicative asymmetric homomorphic encryption, in which $HE_{Enc}(PK_{U_i}, m_1) \odot HE_{Enc}(PK_{U_i}, m_2) = HE_{Enc}(PK_{U_i}, m_1 \times m_2 \mod p)$, where p is a prime modulus.

Moreover, in EDDM all the uploaded files are constructed as a cipher structure form, which consists of a check block (CH), a converting block (CV), an enabling block (EB), and a cipher block (CB). The four blocks of the cipher structure are shown in Fig. 2.

The check block is the identification of a file, and the cloud storage server can identify duplicate cipher structures by it. The enabling block and the cipher block compose a formal digital envelope: the uploaded file is encrypted using a random AES key as the cipher block, and the random AES key is encrypted using the public key of the data owner as the enabling block. Finally, the converting block is the conversion factor, which is used to convert the enabling block and makes the cipher block be shared by each data owner, who has uploaded the duplicate cipher structures.

B. Cipher Structure Construction

To avoid the cloud storage server obtaining the sensitive information from the stored data, users construct their files as the cipher structures which contain the check blocks, the converting blocks, the enabling blocks, and the cipher blocks.

The steps of constructing a cipher structure are as follows: (1) First, a user computes the first hash value of her/his file by the hash function $Hash_1()$ as the check block. (2) Then, she/he randomly selects an AES key, shorter than a prime modulus p, and uses it to encrypt the file as the cipher block. (3) After that, the user encrypts the random AES key using her/his public key by a multiplicative asymmetric homomorphic encryption as the enabling block. (4) Finally, the user computes the second hash value of the file by the hash function $Hash_2()$ and multiplies the second hash value with the random AES key under the modulus p, which is the modulus of the multiplicative asymmetric homomorphic encryption, as the converting block.

After constructing a cipher structure, the user uploads the cipher structure in the cloud storage. The cloud storage server cannot obtain information about the content from the stored cipher structure except the hash value because the content has been encrypted as a formal digital envelope. In addition, the user has to delete the second hash value after constructing.

An example is illustrated below. Assume that there is a user Alice, who has her own public-private key pair (PK_{Alice}, SK_{Alice}) and wants to upload a file F. Before uploading, she constructs the cipher structure as follows:

1) Check Block (CH)
   - Compute the first hash value of the file:
   $$CH = Hash_1(F)$$
2) Cipher Block (CB)
   - Generate a random AES key K_A
   - Encrypt the file using K_A by the AES encryption:
   $$CB = AES_{Enc}(K_A, F)$$
3) Enabling Block (EB)
   - Encrypt K_A using her own public key by the multiplicative asymmetric homomorphic encryption:
   $$EB = HE_{Enc}(PK_{Alice}, K_A)$$
4) Converting Block (CV)
   - Compute the second hash value of the file:
   $$h_2 = Hash_2(F)$$
   - Multiply K_A by h_2 modulo p:
   $$CV = K_A \times h_2 \mod p$$

C. Encrypted Data Deduplication

To avoid storing the same data repeatedly, the cloud storage server stores the whole cipher structure if the content of the cipher structure is uploaded for the first time. Otherwise, the cloud storage server converts the uploaded enabling block (i.e., the encrypted AES key) and stores only the converted enabling block. The data owners can use the converted enabling block to share the same cipher block (i.e., the encrypted file).

The proposed encrypted data deduplication mechanism is described in Algorithm 1. When a cipher structure is uploaded, the cloud storage server checks whether the check block (i.e., the first hash value) is in the hash list. If the check block is not in the hash list, the content of the uploaded cipher structure has not been uploaded. Then, the cloud storage server stores the whole cipher structure and adds the check block into the hash list. Otherwise, the cloud storage server converts the uploaded enabling block: (1) First, the cloud

storage server calculates the modular multiplicative inverse of the uploaded converting block and multiplies it by the first stored converting block under modulus p as the conversion factor. (2) Then, it encrypts the conversion factor by the multiplicative asymmetric homomorphic encryption using the public key of the data owner, who uploads the cipher structure in this session. (3) Finally, the cloud storage server performs the operation $\odot$ on the encrypted conversion factor and the uploaded enabling block to produce the converted enabling block. After converting the uploaded enabling block, the cloud storage server only stores the converted enabling block.

Algorithm 1 Encrypted Data Deduplication

Input: an uploaded cipher structure $\{CH_i,\ CV_i,\ EB_i,\ CB_i\}$ of user U_i

1: **if** $CH_i \notin HL$ **then**
2: Store CV_i, EB_i, and CB_i
3: Create a pointer, which points to the stored cipher structure
4: Add CH_i and the pointer into the hash list HL
5: **else**
6: Retrieve the pointer, P_X, corresponding to CH_i from HL
7: $cf = (P_X \rightarrow CV) \times CV_i^{-1} \mod p$
8: $EB_i' = HE_{Enc}(PK_{U_i},\ cf) \odot EB_i$
9: Store EB_i' and delete the cipher structure uploaded by U_i
10: **end if**

There is a simple example as follows. Assume that Alice and Bob are two users, who have their own public-private key pairs and the same file. First, Alice constructs the file as a cipher structure, $\{CH_A, CV_A, EB_A, CB_A\}$, and uploads it to the cloud storage server. The cloud storage server can verify that the content of the cipher structure is first uploaded by checking the check block (CH_A). Therefore, it adds the hash value into the hash list and stores the other blocks, $\{CV_A, EB_A, CB_A\}$, in the cloud storage. After that, Bob also constructs his cipher structure, $\{CH_B, CV_B, EB_B, CB_B\}$, and uploads it to the cloud storage server. The cloud storage server can verify that the content of the cipher structure has been uploaded by checking the check blocks (CH_B). Therefore, it converts the uploaded enabling block (EB_B):

1) Calculate the modular multiplicative inverse of the converting block, which was uploaded by Bob, and multiply it by the converting block of Alice as the conversion factor:

$$\begin{aligned} cf &= CV_A \times CV_B^{-1} \mod p \\ &= (K_A \times h_2) \times (K_B \times h_2)^{-1} \mod p \\ &= K_A \times K_B^{-1} \mod p \end{aligned}$$

2) Encrypt the conversion factor by the multiplicative asymmetric homomorphic encryption using the public key of Bob:

$$HE_{Enc}(PK_{Bob}, cf)$$

3) Perform the operation $\odot$ on the encrypted conversion factor and the enabling block, which was uploaded by Bob:

$$\begin{aligned} & HE_{Enc}(PK_{Bob}, cf) \odot EB_B \\ = \ & HE_{Enc}(PK_{Bob}, cf \times K_B \mod p) \\ = \ & HE_{Enc}(PK_{Bob}, K_A) \end{aligned}$$

After converting the enabling block of Bob, the cloud storage server eliminates the whole cipher structure, which was uploaded by Bob, and stores only the converted enabling block. When Bob wants to download the stored data, the cloud storage server returns the converted enabling block (EB_B') and the first stored cipher block (CB_A). Bob can decrypt the converted enabling block using his own private key and obtains the AES key K_A, which was selected by Alice. Finally, Bob can decrypt the cipher block (CB_A) using the random AES key K_A and obtains the stored data.

An overview of the proposed mechanism is shown in Fig. 3. In the proposed mechanism, users construct the cipher structures before uploading their files to avoid information leakage. The cloud storage server can store only one copy of the duplicate cipher blocks to avoid storing the same data repeatedly such that the data owners share the stored copy.

IV. Discussions

In this section, we analyze the data confidentiality of the cipher structure which is used in the proposed mechanism and compare the proposed mechanism with the existing data deduplication mechanisms.

A. Security Analysis

The cloud storage server needs to perform data deduplication to avoid the waste of the storage space. For data deduplication, the ciphertexts must be able to be identified whether they correspond to the same plaintext, but it does not mean that the cloud storage server can obtain the contents of the ciphertexts.

We analyze the data confidentiality of the cipher structure (i.e., the check block, the converting block, the enabling block, and the cipher block) as follows:

1) **Check Block**: In the proposed mechanisms, the cloud storage server can identify duplicate cipher blocks, whose contents are the same, by the check blocks (i.e., the first hash values). However, the hash value is irreversible. Therefore, it is hard to retrieve the contents from the check blocks.
2) **Converting Block**: The converting blocks are the second hash values multiplied by random AES keys, which are used to encrypt the files in the cipher blocks, under the modulus p. Due to the security of the cryptographic hash function, it is also hard to retrieve the random AES keys from the converting blocks.
3) **Enabling Block**: The enabling blocks contain the encrypted AES keys which are encrypted by the multiplicative asymmetric homomorphic encryption. It is hard to obtain information about the random AES keys from the enabling blocks.
4) **Cipher Block**: The files are encrypted by the AES encryption in the cipher block. The AES encryption

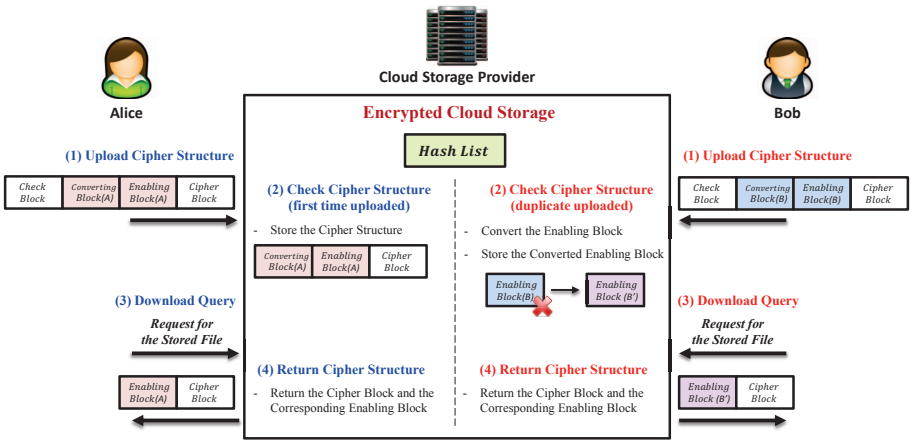

Fig. 3. Overview of the Proposed Mechanism

is a secure encryption algorithm, which is secure enough for U.S. government non-classified data [16]. Hence, it does not exist an adversary who is able to obtain any information about the plaintext from the cipher block in polynomial time.

In short, the files are encrypted by an AES encryption. The AES keys are encrypted using the public keys of data owners based on the multiplicative asymmetric homomorphic encryption in the enabling blocks and multiplied by the second hash values in the converting blocks. There does not exist a polynomial-time adversary who can retrieve the random AES keys or obtain any information about the contents except the hash values.

B. Data Compression

In this subsection, we focus on the performance of data compression. There are four data storage methods as follows:

1) **Stored as Plaintext Form**: All the stored files are not encrypted, and the cloud storage server can use traditional data deduplication to eliminate duplicate files. However, the privacy of the users is not protected because the cloud storage server can obtain the contents of all the stored files.
2) **Stored as Ciphertext Form (I)**: Users encrypt their files using some formal encryption algorithms (e.g., the AES encryption) and store the encrypted files in the cloud storage. The privacy of the users is protected, but the cloud storage server cannot eliminate the duplicate data.
3) **Stored as Ciphertext Form (II)**: Users encrypt their files using convergent encryption and store the encrypted files in cloud storage. By convergent encryption, the cloud storage server can eliminate duplicate ciphertexts by traditional data deduplication. However, the encryption keys are the hash values of the plaintexts not randomly selected, and the encryption is non-ranodomized.
4) **Stored as the Proposed Form**: Each stored file is encrypted using a random key by the AES encryption, and the random key is encrypted by a multiplicative asymmetric homomorphic encryption. The cloud storage server can still eliminate duplicate cipher blocks (i.e., the encrypted files). However, there are some overhead costs.

We take ElGamal's encryption with a 2048-bit modulus to be the multiplicative asymmetric homomorphic encryption in the proposed mechanism, and we adopt the AES encryption with a 256-bit key. TABLE I shows the input length, the output length, and the key length of the hash function and the encryption algorithms. In addition, let $Hash_2$ be a hash function whose output length is 2048 bits. We suppose that there is an uploaded file F, which is 64 KB (65,536 bytes), and the sizes of the stored data in the four storage methods are as follows:

1) **Stored as Plaintext Form**:

$$Plaintext = 65,536\ bytes$$

2) **Stored as Ciphertext Form (I)**:

$$\begin{aligned} Header &= RSA_{Enc}(PK_{U_i},\ K_{AES}) \\ &= \lceil \frac{32}{256} \rceil \times 256 \\ &= 256\ bytes \\ Ciphertext &= AES_{Enc}(K_{AES},\ F) \\ &= \lceil \frac{65,536}{16} \rceil \times 16 \\ &= 65,536\ bytes \end{aligned}$$

3) **Stored as Ciphertext Form (II)**:

$$\begin{aligned} Header &= RSA_{Enc}(PK_{U_i},\ Hash(F)) \\ &= \lceil \frac{64}{256} \rceil \times 256 \\ &= 256\ bytes \\ Ciphertext &= AES_{Enc}(Hash(F),\ F) \\ &= \lceil \frac{65,536}{16} \rceil \times 16 \\ &= 65,536\ bytes \end{aligned}$$

4) **Stored as the Proposed Form**:

TABLE I. INPUT, OUTPUT, AND KEY LENGTHS OF HASH FUNCTION AND ENCRYPTION

Hash Function & Encryption	Input Length	Output Length	Key Length
SHA-2 [17] (Cryptographic Hash Function)	$< 2^{128} - 1$ (bits) $\approx 2^{125}$ (bytes)	512 (bits) = 64 (bytes)	N/A
RSA Encryption [18] (Asymmetric Encryption)	$< (2048 - 1)$ (bits) ≈ 256 (bytes)	2048 (bits) = 256 (bytes)	2048 (bits) = 256 (bytes)
ElGamal Encryption [19] (Homomorphic Encryption)	$< (2048 - 1)$ (bits) ≈ 256 (bytes)	4096 (bits) = 512 (bytes)	2048 (bits) = 256 (bytes)
AES Encryption [16] (Symmetric Encryption)	128 (bits) = 16 (bytes)	128 (bits) = 16 (bytes)	256 (bits) = 32 (bytes)

- Check Block:

$$\begin{aligned} CH &= Hash_1(F) \\ &= 64\ bytes \end{aligned}$$

- Converting Block:

$$\begin{aligned} CV &= K_j \times Hash_2(F) \mod p \\ &= 256\ bytes \end{aligned}$$

- Enabling Block:

$$\begin{aligned} EB &= HE_{Enc}(PK_{U_i}, K_j) \\ &= \lceil \frac{32}{256} \rceil \times 512 \\ &= 512\ bytes \end{aligned}$$

- Cipher Block:

$$\begin{aligned} CB &= AES_{Enc}(K_j, F) \\ &= \lceil \frac{65,536}{16} \rceil \times 16 \\ &= 65,536\ bytes \end{aligned}$$

In EDDM, the size of the cipher structure is 64 KB + (64 + 256 + 512) bytes. When a duplicate cipher structure is uploaded, only the converted enabling block (512 bytes) is stored. Therefore, if n cipher structures, which correspond to the same content, are uploaded to the cloud storage, the cloud storage server only stores a check block, a converting block, a cipher block, and n enabling blocks (64 KB + 320 bytes + 512 $\times n$ bytes).

The comparison of the data compression is showed in TABLE II. Not only does the proposed mechanism protect the uploaded data more secure but also the cloud storage server can perform data deduplication as well.

C. Computation

In this subsection, we focus on the performance of the computation. We compare the existing encrypted data deduplication mechanisms, which use the convergent encryption to encrypt data, with the proposed mechanism. The ciphertext forms are described in Subsection IV-B, and we analyze the computation on the three parts:

1) **Encryption**: Before uploading, users have to encrypt their files as ciphertext form.
2) **Conversion**: When a duplicate ciphertext is uploaded, the cloud storage server will convert the uploaded data such that the data owner can share the stored ciphertext.
3) **Decryption**: Users decrypt the downloaded ciphertexts and obtain the plaintexts.

The comparison of the computation cost is showed in TABLE III. Although the total cost of the proposed mechanism is larger than the existing mechanisms, the greater difference is in the cost of the conversion, which is performed by the cloud storage server. The cloud storage server is more powerful than the client machines and, moreover, it can perform the conversion when it is not busy. Therefore, it does not affect the overall performance.

D. Comparison on Security Issues

The existing encrypted data deduplication mechanisms [7], [8], [13], [14] use the convergent encryption to encrypt data in order to make duplicate ciphertexts identifiable. In those mechanisms, the server can perform the traditional data deduplication to eliminate the duplicate ciphertexts. However, convergent encryption is not a formal encryption algorithm. It uses the hash values of plaintexts to be the encryption keys and encrypts the plaintexts by a non-randomized encryption.

In contrast, the proposed mechanism improves the data confidentiality. The uploaded data are encrypted by formal encryptions (e.g., the AES encryption), and the encryption keys are randomly selected, which are independent of the plaintexts. The cloud storage server can eliminate the duplicate cipher blocks (i.e., the encrypted files) but cannot obtain any information about the plaintexts except the hash values. TABLE IV shows the comparison between the proposed mechanism and the existing mechanisms.

V. CONCLUSION

In this manuscript, we propose a feasible encrypted data deduplication mechanism in which all the data are stored as the cipher structure form, which consists of a check block, a converting block, an enabling block, and a cipher block. The cloud storage server can identify the duplicate cipher structures by the check blocks and convert the duplicate enabling blocks (i.e., the encrypted AES keys). Each data owner of the duplicate cipher structures, which correspond to the same plaintext, can share the same copy of the cipher blocks (i.e., the encrypted file). Therefore, the cloud storage server stores only one copy of the duplicate cipher block, thus avoiding the waste of storage space.

As compared with the existing encrypted data deduplication mechanisms, the proposed mechanism improves the security of the ciphertexts where the encryption is randomized, and the encryption keys are randomly selected, not the hash

TABLE II. COMPARISON OF DATA COMPRESSION

	Size of n Copies	Storage Consumption	Data Compression Ratio [♯1]
Stored as Plaintext Form	(64 KB)$\times n$	64 KB	$\frac{1}{n}$ [♯3]
Stored as Ciphertext Form (I)	(64 KB)$\times n$ + (256 bytes) $\times n$	(64 KB)$\times n$ + (256 bytes) $\times n$	1 [♯3]
Stored as Ciphertext Form (II)	(64 KB)$\times n$ + (256 bytes) $\times n$	64 KB + (256 bytes) $\times n$	$\approx \frac{1}{n} + 0.004$ [♯4]
Stored as the Proposed Form	(64 KB + 320 bytes) $\times n$ + (512 bytes) $\times n$	64 KB + 320 bytes + (512 bytes) $\times n$	$\approx \frac{1}{n} + 0.008$ [♯5]

[♯1] Data Compression Ratio = $\frac{the\ compressed\ size}{the\ uncompressed\ size}$. The compressed size is the size after using data deduplication in the cloud storage. The uncompressed size is the size of the plaintexts.
[♯2] Data Compression Ratio = $\frac{65,536}{65,536\times n} = \frac{1}{n}$
[♯3] Data Compression Ratio = $\frac{65,536\times n}{65,536\times n} = 1$
[♯4] Data Compression Ratio = $\frac{65,536+256\times n}{65,536\times n} = \frac{65,536}{65,536\times n} + \frac{256}{65,536} \approx \frac{1}{n} + 0.004$
[♯5] Data Compression Ratio = $\frac{65,536+320+512\times n}{65,536\times n} = \frac{65,856}{65,536\times n} + \frac{512}{65,536} \approx \frac{1}{n} + 0.008$

TABLE III. COMPARISON OF COMPUTATION

	Encryption	Decryption	Conversion
Existing Mechanisms	$T_H + T_{AE} + T_{SE}$	$T_{AE} + T_{SE}$	0
Proposed Mechanism	$2T_H + T_{HE} + T_{SE} + t_m$	$T_{HE} + T_{SE}$	$T_{HE} + t_m + t_{he}$

t_m: the time cost of a modular multiplication
t_{he}: the time cost of a multiplicative asymmetric homomorphic operation $\odot$
T_H: the time cost of a hash
T_{AE}: the time cost of the encryption and decryption of a symmetric encryption
T_{AE}: the time cost of the encryption and decryption of a asymmetric encryption
T_{HE}: the time cost of the encryption and decryption of a multiplicative asymmetric homomorphic encryption

values of the plaintexts. As a result, the cloud storage server cannot obtain any information about the stored data except the hash values.

ACKNOWLEDGMENT

This work was supported in part by the Taiwanese Ministry of Science and Technology under Grant MOST 103-2221-E-110-057 and by the NSYSU and the Taiwanese Ministry of Education through the Aim for the Top University Plan.

REFERENCES

[1] J. Harauz, L. M. Kaufman, and B. Potter, "Data security in the world of cloud computing," *IEEE Security and Privacy*, vol. 7, no. 4, pp. 61–64, 2009.

[2] D. Zissis and D. Lekkas, "Addressing cloud computing security issues," *Future Generation Computer Systems*, vol. 28, no. 3, pp. 583–592, 2012.

[3] D. Geer, "Reducing the storage burden via data deduplication," *IEEE Computer*, vol. 41, no. 12, pp. 15–17, 2008.

[4] Q. He, Z. Li, and X. Zhang, "Data deduplication techniques," in *Proc. 2010 Int. Conf. on Future Information Technology and Management Engineering (FITME 2010)*, 2010, pp. 430–433.

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

[6] J. Xu, E. Chang, and J. Zhou, "Leakage-resilient client-side deduplication of encrypted data in cloud storage," in *Cryptology ePrint Archive: Report 2011/538*, 2011, http://eprint.iacr.org/2011/538.

[7] P. Anderson and L. Zhang, "Fast and secure laptop backups with encrypted de-duplication," in *Proc. 24th Int. Conf. on Large Installation System Administration (LISA 2010)*, 2010, pp. 1–8.

[8] J. R. Douceur, A. Adya, W. J. Bolosky, D. Simon, and M. Theimer, "Reclaiming space from deduplicate files in a serverless distributed file system," in *Proc. 22nd Int. Conf. on Distributed Computing Systems (ICDCS 2002)*, 2002, pp. 617–624.

[9] K. Jin and E. L. Miller, "The effectiveness of deduplication on virtual machine disk images," in *Proc. SYSTOR 2009: The Israeli Experimental Systems Conference*, 2009, p. Article No. 7.

[10] S. Lee and D. Choi, "Privacy-preserving cross-user source-based data deduplication in cloud storage," in *Proc. IEEE International Conference on ICT Convergence (ICTC 2012)*, 2012, pp. 329–330.

[11] C. Liu, D. Ju, Y. Gu, Y. Zhang, D. Wang, and D. Du, "Semantic data de-duplication for archival storage systems," in *Proc. 13th IEEE Asia-Pacific Computer Systems Architecture Conference (ACSAC 2008)*, 2008, pp. 1–9.

[12] A. Muthitacharoen, B. Chen, and D. Mazires, "A low-bandwidth network file system," in *Proc. 18th ACM Symposium on Operating Systems Principles (SOSP2011)*, 2011, p. 174187.

[13] M. W. Storer, K. Greenan, D. D. Long, and E. L. Miller, "Secure data deduplication," in *Proc. 4th ACM International Workshop on Storage Security and Survivability*, 2008, pp. 1–10.

[14] C. Wang, Z. Qin, J. Peng, and J. Wang, "A novel encryption scheme for data deduplication system," in *Proc. IEEE Int. Conf. on Communications, Circuits and Systems (ICCCAS 2010)*, 2010, pp. 265–269.

[15] J. Pettitt, ""RE: Hash of plaintext as key?", cypherpunks mailing list, http://cypherpunks.venona.com/date/1996/02/msg02013.html."

[16] J. Daemen and V. Rijmen, "The advanced encryption standard (AES)," *United States National Institute of Standards and Technology (NIST)*, vol. Federal Information Processing Standards Publication 197, November 26 2001, http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf.

[17] N. S. A. (NSA), "Secure hash standard (SHS)," *United States National Institute of Standards and Technology (NIST)*, vol. Federal Information Processing Standards Publication 180-4, March 2012, http://csrc.nist.gov/publications/fips/fips180-4/fips-180-4.pdf.

[18] R. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Communications of the ACM*, vol. 21, no. 2, p. 120126, 1978, http://people.csail.mit.edu/rivest/Rsapaper.pdf.

[19] T. Elgamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," *IEEE Trans. on Information Theory*, vol. 31, no. 4, pp. 469–472, 1985.

TABLE IV. COMPARISON OF DATA DEDUPLICATION

Mechanisms	Deduplication Approach	Encryption Algorithm	Encryption Key	Key Management
Douceur *et al.*'s Mechanism [8]	Target-Based	Non-Randomized Encryption	Hash Value of Plaintext	Block Header ♯1
Storer *et al.*'s Mechanism [13]	Target-Based	Non-Randomized Encryption	Hash Value of Plaintext	User ♯2
Wang *et al.*'s Mechanism [14]	Target-Based	Non-Randomized Encryption	Hash Value of Plaintext	Third Party ♯3
Anderson *et al.*'s Mechanism [7]	Target-Based	Non-Randomized Encryption	Hash Value of Plaintext	Block Header ♯1
The Proposed Mechanism	Target-Based / Source-Based	Randomized Encryption	Random Value	Block Header ♯1

♯1 The encryption key is encrypted using the user's own public key as a block, and the block is also uploaded to the cloud storage with the ciphertext.
♯2 Users have to retain the encryption keys by themselves.
♯3 There is a third parity who is responsible for maintaining the information which users can use to rebuild and decrypt the stored ciphertexts.