Secure Cloud Data Deduplication with Efficient Re-encryption

Haoran Yuan, Xiaofeng Chen, *Senior Member, IEEE*, Jin Li, Tao Jiang, Jianfeng Wang, and Robert H. Deng, *Fellow, IEEE*

Abstract—Data deduplication technique has been widely adopted by commercial cloud storage providers, which is both important and necessary in coping with the explosive growth of data. To further protect the security of users' sensitive data in the outsourced storage mode, many secure data deduplication schemes have been designed and applied in various scenarios. Among these schemes, secure and efficient re-encryption for encrypted data deduplication attracted the attention of many scholars, and many solutions have been designed to support dynamic ownership management. In this paper, we focus on the re-encryption deduplication storage system and show that the recently designed lightweight rekeying-aware encrypted deduplication scheme (REED) is vulnerable to an attack which we call it stub-reserved attack. Furthermore, we propose a secure data deduplication scheme with efficient re-encryption based on the convergent all-or-nothing transform (CAONT) and randomly sampled bits from the Bloom filter. Due to the intrinsic property of one-way hash function, our scheme can resist the stub-reserved attack and guarantee the data privacy of data owners' sensitive data. Moreover, instead of re-encrypting the entire package, data owners are only required to re-encrypt a small part of it through the CAONT, thereby effectively reducing the computation overhead of the system. Finally, security analysis and experimental results show that our scheme is secure and efficient in re-encryption.

Index Terms—Re-encryption, Data deduplication, User joining, User revocation.

1 Introduction

With the rapid development of cloud storage, more and more individuals and enterprises tend to outsource their sensitive data to remote cloud service providers in a pay-per-use manner [1], [2], [3], [4], [5]. According to the study from Internet Data Center (IDC) sponsored by Dell EMC, the digital universe is doubling in size every two years and the volume of the universe data is expected to reach 44 zettabytes (ZB) or 44 trillion gigabytes (GB) in 2020 (more than 5200 gigabytes for each man, woman, and child) [6]. However, the growth of data puts heavy pressures on cloud service providers. To cope with it, a straightforward method is to require cloud service providers continuously increasing the capacity of storage space, so as to meet users' requirements for high-quality storage services.

However, cloud service providers may store plentiful and repetitive data (such as movies, music and genome data), which inevitably incurs a mass of redundant storage and backup space, consequently to cost a vast amount of computing and management overhead during its whole life cycle. To solve this problem, Bolosky *et al*. first proposed the technique of data deduplication [7], which decreases the redundant storage space and bandwidth by eliminating duplicate copies and only storing one copy of them. Nowadays, data deduplication techniques have been widely deployed by cloud service providers, such as Dropbox [8], Google Drive [9] and Memopal [10]. Researches [11], [12] have shown that most of the genome data ($\geq 83\%$) and disk ($\geq 90\%$) of business applications can be reduced by exploiting data deduplication technique. While the technique of data deduplication has plentiful advantages, there are still some security challenges that need to be addressed. In particular, the cloud service providers are often assumed to be not fully trusted, which may try to infer and analyze the outsourced data [13], [14], [15], [16]. To protect the confidentiality of their sensitive data, cloud users generally encrypt their data before outsourcing them to cloud service providers. However, different users encrypt the same data with their private keys, which leads the same data to output different ciphertexts, and makes the function of data deduplication unachievable. Douceur *et al*. [17] proposed the first feasible solution to protect the confidentiality of data and achieve deduplication on ciphertexts. However, the cloud user encrypts sensitive data with a convergent key, which is derived from the hash value of the data and unchanged. It will lead the revoked cloud user to access the sensitive data through the reserved convergent key.

User revocation is a severe security problem in the cloud environment. We take the case in genome research as an example to illustrate this point. Considering the enormous volume of genome datasets, genome researchers tend to use the cloud to store the genome data [18]. Google Genomics [19] and Amazon [20] have deployed specific platforms for managing and analyzing genome data. However, some

- *H. Yuan, X. Chen, T. Jiang and J. Wang are with the State Key Laboratory of Integrated Service Networks, Xidian University, Xi'an 710071, P.R. China and the State Key Laboratory of Cryptology, P.O. Box 5159, Beijing, 100878, P.R. China.*
  E-mail: hryuan1@163.com xfchen@xidian.edu.cn taojiang@xidian.edu.cn jfwang@xidian.edu.cn
- *J. Li is with the School of Computer Science, Guangzhou University, Guangzhou, PR China.*
  E-mail: lijin@gzhu.edu.cn
- *R. H. Deng is with the School of Information Systems, Singapore Management University, Singapore.*
  E-mail: robertdeng@smu.edu.sg

sensitive genome data produced by disease sequencing projects must be protected. For example, when a researcher is no longer a member of the genome project, he will be prohibited from accessing the genome datasets. This problem has been addressed by using techniques such as re-encryption and group key distribution [21], [22], [23]. By using symmetric encryption (such as AES-128 or AES-256) to re-encrypt the sensitive data and distribute group key for group users, those schemes can support user joining and user revocation. Although the re-encryption scheme uses a new encryption key to encrypt the entire message to protect the data confidentiality, it will result in a waste of excessive computation overhead. William *et al*. [24] provided evidence that, in situations involving even a minimal amount of policy dynamism, the cryptographic enforcement of access controls is likely to carry prohibitive costs.

Recently, Li *et al*. [25], [26] proposed a rekeying-aware encrypted deduplication storage system (REED), which supports a lightweight re-encryption. Instead of re-encrypting the entire package, data owners are only required to re-encrypt a small part of it through the CAONT, thereby effectively reducing the computation overhead of the system. However, we point that the REED is vulnerable to the stub-reserved attack, which will be described in Section 3.2. In short, if a revoked cloud user keeps the last bytes of a package as $stub$ package, he can use the reserved $stub$ package and $trimmed$ package (downloaded from the cloud service providers) to recover the plaintext. Therefore, existing proposed schemes cannot well support secure dynamic ownership management of cloud users and efficient re-encryption.

Our Contribution. In this paper, we further study the above problems of secure and efficient re-encryption for deduplication storage. Our contributions are three folds:

- We point out a security weakness of the enhanced encryption of REED scheme [25], [26]. That is, this scheme is vulnerable to the so-called stub-reserved attack proposed in this paper.
- We propose a location selection method based on Bloom filter and a secure data deduplication scheme with efficient re-encryption. By using the symmetric encryption and this new location selection method, the revoked cloud user cannot obtain the sensitive data from the data owner. Thus data privacy is ensured. Moreover, instead of re-encrypting the entire package, data owners are only required to re-encrypt a small part of it through the CAONT, thereby effectively reducing the computation overhead of the scheme.
- We provide security analysis and performance evaluation of our scheme, and the results show that our scheme is secure and efficient.

2 Preliminaries

In this section, we first present the definitions and properties of all-or-nothing transform (AONT) [27] and convergent all-or-nothing transform (CAONT) [28]. Then, we review the concept of the ciphertext-policy attribute-based encryption (CP-ABE) and Bloom filter.

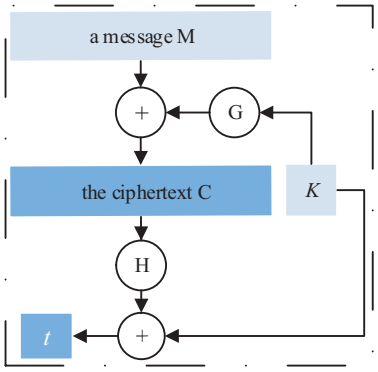


Fig. 1. All-or-nothing transform.

2.1 Convergent All-or-nothing Transform

All-or-nothing transform (AONT) is an unkeyed and randomized encryption mode, which has the property that one must decrypt the entire ciphertext before one can determine even one message block [27]. AONT is defined as follows.

AONT transforms a message M into a package (C, t), where C is called $head$ and t is called $tail$. In particular, a user first chooses a random key K and generates a pseudo-random mask $\mathsf{G}(K) = \mathsf{E}(K, P)$, where $\mathsf{E}(\cdot)$ denotes a symmetric key encryption algorithm (e.g., AES-256) and P denotes a public block with the same size as message M. Then, the user computes $C = M \oplus \mathsf{G}(K)$ and $t = \mathsf{H}(C) \oplus K$, where $\oplus$ is the exclusive-or (XOR) operation and $\mathsf{H}(\cdot)$ denotes a hash function (e.g., SHA-256). It should be noted that the resulting package (C, t) is longer than message M. To recover the original message M, a user first uses C and t to compute $K = \mathsf{H}(C) \oplus t$. Then, the user uses C and K to compute $M = C \oplus \mathsf{E}(K, P)$. The workflow of the AONT is shown in Fig. 1.

In the original AONT scheme, it randomly chooses a key K to construct a package. Thus, the original AONT scheme hinders the implementation of data deduplication. To solve this problem, Li *et al*. proposed the CAONT scheme [28]. CAONT derives the deterministic key from the identical message. By using the deterministic key to construct a package, the same message always generates the same package. Especially, CAONT follows the same paradigm of AONT by replacing the encryption key with a deterministic hash value $k = \mathsf{H}(M)$. This makes data deduplication over encrypted data plausible. Furthermore, CAONT allows integrity verifying without padding. The integrity can be checked by computing the hash of message M and checking whether it equals k.

To prove the security of the AONT scheme, Victor Boyko defined the non-adaptive indistinguishability in [29]. We review the scenario and definition of non-adaptive indistinguishability as follows. Let L be an arbitrary set of l bit positions. The adversary runs in two stages:

- Find stage: The adversary is given L and access to random oracle $\mathcal{O}$. He outputs $x_0, x_1 \in \{0,1\}^n$ and $c_f \in \{0,1\}^*$.
- Guess stage: The adversary is given c_f and, for random bit b, $\mathrm{AONT}^{\mathcal{O}}(x_b)$ with bit positions L missing. The adversary has access to $\mathcal{O}$ and guesses b.

Note that x_0 and x_1 may be included in c_f. We may view c_f as the saved state of the adversary at the end of the

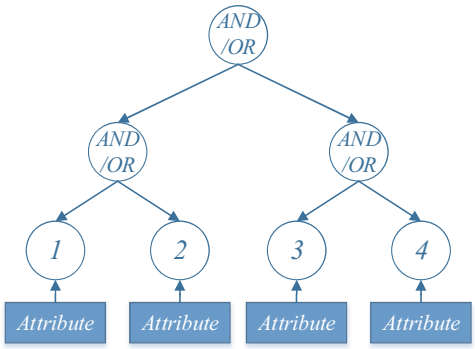


Fig. 2. Access tree for CP-ABE.

find stage. We want the adversary's probability of correctly guessing b to be as close as possible to $\frac{1}{2}$. The definition is as follows:

Definition 1. *(Non-adaptive indistinguishability). Let AONT be a randomized transform mapping n-bit messages to n'-bit outputs and using random oracle $\mathcal{O}$. Let l be between 1 and n'. An adversary A is said to succeed in $(T, q_{\mathcal{O}}, \varepsilon)$-distinguishing AONT with l missing bits if there exist $L \in \{n', l\}$ such that* $\Pr[\mathcal{O} \xleftarrow{R} \Omega; (x_0, x_1, c_f) \xleftarrow{R} A^{\mathcal{O}}(L, \text{find}); b \xleftarrow{R} \{0,1\}; y \xleftarrow{R} AONT^{\mathcal{O}}(x_b) : A^{\mathcal{O}}(h_{n',L(y),c_f}, \text{guess}) = b] \geqslant 1/2 + \varepsilon$*, and, moreover, in the experiment above, A runs for at most T steps, and makes at most $q_{\mathcal{O}}$ queries to $\mathcal{O}$.*

To ensure the security of ANOT, for any given an L, the advantage of an arbitrary adversary to break the Non-adaptive indistinguishability should be less than ε.

2.2 Ciphertext-Policy Attribute-Based Encryption

Ciphertext-policy attribute-based encryption (CP-ABE) is a cryptographic encryption algorithm, which enables data owners to define an access policy over user attributes and encrypt data under the access policy with the corresponding public key components [30]. If and only if the users' attributes match the corresponding access policy, the user can decrypt the corresponding ciphertext. In CP-ABE, each policy is represented in an access tree, in which each non-leaf node of the tree represents a Boolean gate (e.g., AND or OR), while each leaf node represents an attribute that defines user property (e.g., ages, genders, departments, etc.). Each user is given a private key that corresponds to a set of attributes. If and only if the user's attributes satisfy the access tree, his private key can decrypt the ciphertext. The access tree of CP-ABE is shown in Fig. 2.

In this paper, we treat each attribute as a unique identifier of each cloud user. We issue each cloud user with a CP-ABE private key, which is related to the identity. The policy of each file as an access tree and the identities of all authorized cloud users are connected with the OR gate. Therefore, any authorized cloud user can decrypt the ciphertext and get the original message.

2.3 Bloom filter

The Bloom filter is a simple space-efficient data structure for checking whether an element belongs to a set [31], which has been very popular in practical applications. A Bloom filter sets a bit array of n bits and k independent hash functions defined as follows: $h_i : \{0,1\}^* \rightarrow [1,n], i \in [1,k]$. These hash functions map each element to a random number uniformed in $\{1, ..., n\}$. In the initialization stage, all bits of the array are set to 0. To add an element x to the set, it computes k array locations with the k hash functions, and sets all the locations of $h_i(x)$ as 1. To check whether the element x is in the set S, a user just needs to compute $h_i(x)$ for reconstructing all the array locations. Thus, if one of the array locations is 0, we can make sure that the element x is not in the set S. However, it is not certain whether the element is in the set even all the locations of $h_i(x)$ are 1, since there exist $h_i(x) = h_i(y)$ (i.e., collision of the hash function) for some $y \neq x$.

3 Analysis of REED Scheme

Li *et al*. proposed a novel rekeying-aware encrypted deduplication storage system, called REED [25], [26]. In this section, we first give an overview of the REED scheme. Then, we point out the security weakness in this construction.

3.1 Review of REED Scheme

Based on the CAONT mechanism, Li *et al*. proposed a rekeying-aware encrypted deduplication storage system, called REED. Due to the feature of CAONT, it is computationally infeasible to recover the original message without getting the entire ciphertext. Hence, instead of re-encrypting the entire package, data owners are only required to re-encrypt a small part of it through the CAONT, thereby effectively reducing the computation overhead of the system.

Enhanced encryption of REED scheme: To enhance the security of the basic scheme, the authors proposed an enhanced scheme. The workflow of the enhanced scheme is shown in Fig. 3. We review the enhanced scheme as follows. If a cloud user wants to upload a message M, the cloud user first uses the message-locked encryption (MLE) key K_M to encrypt the message M by the traditional MLE scheme and get the ciphertext $C_1 = \mathsf{E}(K_M, M)$, where $\mathsf{E}(\cdot)$ is the encryption function. Then, the cloud user transforms the $K_M||C_1$ by the original CAONT, where "||" denotes the concatenation. Different from the basic scheme, the enhanced scheme uses the hash key $h = \mathsf{H}(C_1||K_M)$ as the input of pseudo-random mask G. The cloud user computes $\mathsf{G}(h) = \mathsf{E}(h, S)$, where S is a publicly known block with the same size as $C_1||K_M$. The package head is $C_2 = (C_1||K_M) \oplus \mathsf{G}(h)$. To generate the package tail t, the cloud user first divides C_2 into a set of fixed-sized pieces, each with the same size as h. Then, the cloud user performs XOR operation of all the pieces as well as h to generate the package tail t. Without getting the entire message C_2, the result of self-XOR cannot be predicted. Finally, the cloud user trims the last few bytes (e.g., 64 bytes) from (C_2, t) as the *stub* package and leaves the remaining part of the package as the *trimmed* package. For preventing the revoked cloud user from recovering the original message, the data owner uses the file key to re-encrypt the *stub* package and generate the *stub'* package. The file key is distributed by using the CP-ABE algorithm. The cloud user who belongs to the group can get the file key. Furthermore, the data owner

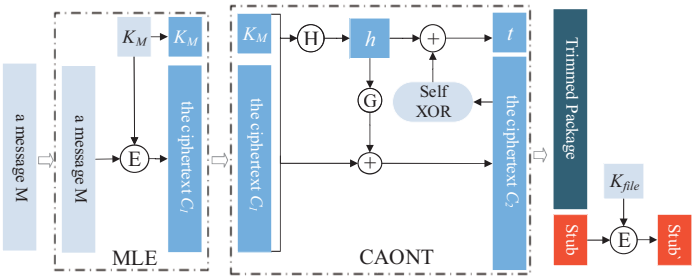


Fig. 3. Enhanced encryption of REED.

only needs to re-encrypt the $stub$ package to generate the $stub'$ package. Finally, the cloud user only uploads the $stub'$ and $trimmed$ packages to the cloud service provider.

The procedure of message reconstruction is as follows. Firstly, the cloud user uses the file key to decrypt the $stub'$ package and get the $stub$ package. For reconstructing (C_1, K_M) from the $trimmed$ and $stub$ packages, the cloud user divides C_2 into fixed-size pieces and recovers h by performing XOR operation of all the pieces and t. Then, the package $C_1||K_M$ is generated by comparing $C_1||K_M = C_2 \oplus \mathsf{G}(h)$ and the integrity is checked by comparing $\mathsf{H}(C_1||K_M)$ and h. Finally, the cloud user computes $M = \mathsf{D}(K_M, C_1)$, where $\mathsf{D}(\cdot)$ is the decryption function.

3.2 Stub-reserved Attack

We argue that the REED scheme suffers from the following attack.

We assume a rational attacker model that a malicious user wants to access the original message after key revocation by minimizing the local storage effort. This attack can be launched by a revoked cloud user who aims to recover the original message without data ownership. The attack is based on the fact that a cloud user can use the file key to decrypt the $stub'$ package and get the $stub$ package before the cloud user leaves the group. After being revoked from the group, the revoked cloud user can use the reserved $stub$ package and $trimmed$ package (corresponding to the $stub$ package) downloaded from the cloud service provider to reconstruct the original message. Because the $stub$ package only occupies 0.39% of the original message, the revoked cloud user can keep a small part of data for recovering the original message. The workflows of the stub-reserved attack are described as follows.

1) Before a cloud user leaves the group of the message M, the cloud user uses the file key K_{file} to decrypt the $stub'$ package and get the $stub$ package. The $stub$ package is stored in local storage and used to recover the original message.
2) After being revoked from the group, the cloud user concatenates $trimmed$ package (downloaded from the cloud service providers) and reserved $stub$ package to recover (C_2, t) package (because the $stub$ package is the last few bytes of (C_2, t) and the $trimmed$ package is the remaining part of the package of (C_2, t)).
3) The revoked cloud user divides C_2 into fixed-size pieces and computes h by performing XOR operation of those pieces and t. The package $C_1||K_M$ is recovered by computing $C_1||K_M = C_2 \oplus \mathsf{G}(h)$. Then, the cloud user splits the package $C_1||K_M$ to the ciphertext C_1 and the MLE key K_M.
4) Finally, the cloud user can use K_M to decrypt C_1 and get the original message $M = D(K_M, C_1)$. Visually, the reversed $stub$ package can be regarded as the "big key", which is longer than the file key but significantly shorter than the original message.

We analyze why the REED scheme suffers from the stub-reserved attack. The main reason is that the small part of the ciphertext (which is used for re-encryption) never changes, this makes the revoked cloud user can predict which small part of the ciphertext (that is $stub$ package) will be re-encrypted. Then, the revoked cloud user can recover the original message by using the reserved $stub$ package. In detail, because the REED scheme uses the CAONT mechanism, it is computationally infeasible to recover the original message without getting the entire ciphertext. Then, the data owners only need to re-encrypt a small part of the entire ciphertext for saving computation overhead. However, if the sites of $stub$ package never change, the revoked cloud user can recover the original message by using the reserved $stub$ package rather than the file key. Although the $stub$ package is longer than the file key, it still far less than the plaintext (0.78% for an 8 KB chunk and 0.39% for a 16 KB chunk).

Remark 1. *In order to deal with this problem, a straightforward solution is that the data owner randomly chooses a small part of the ciphertext as the stub package. This makes the revoked cloud user cannot predict which part of ciphertext will be chosen in the next re-encryption. Thus, the revoked cloud user needs to preserve the entire stub and trimmed packages to recover the original message. Because the stub and trimmed packages are longer than the original message, it is better to preserve the original message. This solution can avoid the stub-reserved attack. However, the revoked cloud user still can keep the h before he leaves the group. Thus, the revoked cloud user can use h as the input of pseudo-random mask* G *to generate* $\mathsf{G}(h)$*. The revoked cloud user performs XOR operation of* $\mathsf{G}(h)$ *and package trimmed to extract a majority part of the chunk. Therefore, the privacy of the data owners' sensitive data is compromised.*

4 Models and Security Goals

4.1 System Model

In this paper, we propose a secure cloud data deduplication scheme with efficient re-encryption. Our scheme is designed for enterprise or user groups in which multiple users want to outsource the data to a remote cloud service provider. The cloud service provider can conduct deduplication on ciphertexts and save abundant storage overhead. The system of our scheme contains three entities: cloud user, key server and cloud service provider (CSP). The system model of our scheme is shown in Fig. 4.

Data deduplication schemes can be categorized into file-level and block-level deduplication. This paper focuses on file-level deduplication, which divides file data into the fixed-size chunk. In this paper, we use the terms "chunks" and "message" interchangeably.

- Cloud user: A cloud user is an entity who wants to outsource sensitive data to the CSP and access

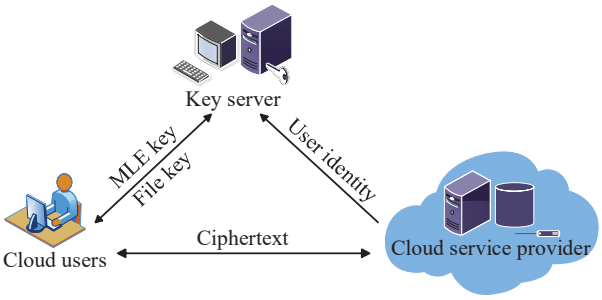


Fig. 4. The cloud storage model.

data later. After uploading the data, the cloud user removes M for saving storage space.

- Key server: The key server is a CP-ABE key server, which helps the cloud user to distribute the file key. The key server is also used to generate the MLE encryption key. Cloud users obtain the encryption key from the key server via an oblivious PRF protocol. We assume that the key server is trusted worthy and the communication channel is encrypted and authenticated.
- CSP: The CSP is an entity that provides storage services, which is responsible for deleting redundant data and only storing one copy of them. The CSP is assumed to be honest-but-curious. That is, it will execute the protocol honestly, but also is curious about the contents of stored data. The CSP should not be accessible to the plaintext. In particular, as claimed in [32], we also hold the same assumption that the CSP would not store all the history version of the $gathered$ package and $remainder$ package[1]. Our scheme provides strong economic incentives for the CSP to store only the newest $gathered$ and $remainder$ packages: A misbehaving CSP storing all the history version of the $gathered$ package and $remainder$ package would have to multiply its storage cost.

4.2 Threat Model

We assume a rational adversary model that an adversary wants to access the original message by minimizing the local storage effort. Otherwise, if the stored data is larger than the original plaintext, it is better to store the original plaintext. The attacker capability is described as following. First, it can compromise the CSP and access all the stored chunks. Second, it can collude with a subset of unauthorized or revoked cloud users, and attempt to access the original message that is beyond the access scope of the colluded users. Furthermore, it can monitor the activities of the cloud user, identify the MLE keys returned by the key server, and extract the message owned by the monitored cloud user.

We assume that our threat model under the following assumptions. First, we assume that the communication between a cloud user and the key server is encrypted and authenticated (e.g., using SSL/TLS). Thus, any eavesdropping activity can be resisted. Second, because our scheme adopts the oblivious key generation scheme, the key server cannot infer the fingerprint information and learn the message content. Third, we assume that the key server is trusted. The adversary cannot collude with the key server. Besides, the key server can rate-limit the query rate of each cloud user. The on-line brute force attacks from a compromised cloud user can be avoided. Since our scheme performs server-side deduplication, any side channel is not introduced in our scheme [33].

1. If CSP keeps all the history version of the ciphertext and the revoked cloud user keeps the corresponding key. Once the CSP and revoked cloud user collude, the revoked cloud user always can decrypt the ciphertext in almost all of the re-encryption schemes.

4.3 Security Goals

The security goals of our scheme include three folds. First, our scheme ensures *integrity*, such that when a cloud user downloads a package from the CSP, the cloud user can verify whether it has been corrupted. Second, our scheme ensures *confidentiality*. The adversary, unauthorized cloud user and revoked cloud user should not obtain the plaintext. More precisely, before uploading the data, a cloud user should be assumed to be unauthorized, although the cloud user possesses the data. Thus, the cloud user should be prevented from accessing the plaintext of the outsourced data. After being revoked from the group, the cloud user is also assumed to be unauthorized. Thus, the revoked cloud user should also be prevented from accessing the outsourced data. Finally, our scheme prevents the *stub-reserved attack*.

5 Scheme Construction

In this section, we first introduce the main idea of our scheme. Then, we introduce the Bloom filter-based location selection method. Finally, we present our scheme in detail.

5.1 Main Idea

The CAONT mechanism has the property that one must decrypt the entire ciphertext before one can determine even one message block. If a data owner transforms a message into the t and C_2 packages by using CAONT, the data owner is not required to re-encrypt the entire package but only a small part of the package C_2, which saves excessive computation overhead. However, to protect the data privacy of data owners' sensitive data, not only do we need to prevent the revoked cloud user from accessing the original message, but also need to prevent the CSP from accessing the original message. In order to solve the above problems, we propose a Bloom filter-based location selection method and a secure data deduplication scheme with efficient re-encryption. For protecting data privacy, a data owner generates a two-part $gathered$ package as follows. Firstly, the data owner trims the last 256-bit from the package C_2 as part-two of the $gathered$ package. Secondly, the data owner uses the Bloom filter-based location selection method to select the random 256 locations of the package C_2 (after trimming 256-bit). Then, the data owner gathers the randomly selected 256-bit as part-one of the $gathered$ package. The package t and a large part of the remained package C_2 make up the $remainder$ package. Finally, the data owner uses the file key to re-encrypt the $gathered$ package and generate the $gathered'$ package. The data owner uploads the $gathered'$ and $remainder$ packages to the CSP. The CP-ABE key server

encrypts the file key by using CP-ABE based on the policy of the file, which allows authorized cloud users to get the file key. Due to the property of the hash function, a revoked cloud user is hard to predict the correct locations in the message without a new file key. In addition, the CSP is hard to get the last 256-bit of the message. Then, it is difficult to reconstruct the original message. Therefore, our scheme is secure against the stub-reserved attack and prevents the CSP from accessing the plaintext.

In order to achieve data deduplication, our scheme enables the different cloud users with the same file always generate the same ciphertext. The process of encryption is described as follows. A cloud user first uses the MLE algorithm to encrypt the message M and get the ciphertext C_1. Then, the cloud user transforms the ciphertext C_1 into the t and C_2 packages by using CAONT. Secondly, the cloud user concatenates the randomly selected 256-bit of package C_2 by using the Bloom filter-based location selection method and the last 256-bit from the package C_2 as $gathered$ package. The package t and a large part of the remained package C_2 are concatenated as the $remainder$ package. Finally, the cloud user uses the file key[2] to re-encrypt the $gathered$ package and generate the $gathered'$ package. After that, the cloud user uploads the $gathered'$ and $remainder$ packages to the CSP. In this way, our scheme allows different cloud users with the same data always to generate the same $remainder$ and $gathered'$ packages. Since the different cloud users outsource the same $remainder$ and $gathered'$ packages to the CSP, the CSP can remove the duplicate packages and only keep one copy of them. Thus, our scheme can achieve ciphertext deduplication.

5.2 Bloom Filter-Based Location Selection Method

As we analyzed in Section 3.2, we need to random choose location from package C_2 to generate the $gathered$ package. A straightforward solution is to define 256 hash functions for selecting random locations. In detail, a cloud user inputs the secret key into 256 hash functions to generate 256 random locations. Then, the cloud user sorts those 256 randomly chosen locations in ascending order. Finally, the cloud user gathers all of the bit values corresponding to these locations from package C_2 and concatenates the last 256 bits of C_2 as the $gathered$ package. This straightforward solution can prevent the stub-reserved attack. However, it has the following problems. First, the cloud user needs to store 256 hash values, which increases the storage cost. Second, the cloud user needs to sort those 256 randomly chosen locations in ascending order, which increases the computational cost.

In order to solve the above problems, we propose a Bloom filter-based location selection method. The main idea of the Bloom filter-based location selection method is to adopt the bit array of Bloom filter to record the selected locations, which can reduce storage and computation overhead. In detail, we set a bit array of n bits and 256 independent hash functions defined as follows: $h_i = \mathsf{H}(key, i) : \{0,1\}^* \rightarrow [1, n], i \in [1, 256]$. These hash functions map each element to a random number uniform over $\{1, ..., n\}$. In the initialization stage, all bits of the bit array are set to 0. To select 256 random array locations, a cloud user randomly chooses a key and computes $h_i = \mathsf{H}(key, i) : \{0,1\}^* \rightarrow [1, n]$ from $i = 1$ to $i = 256$. Then, all 256 locations of the bit array are set to 1. The location of 1 in the bit array represents the 256 randomly selected location. Different from the Bloom filter, if two hash functions generate the same hash value, the second hash function re-computes a new hash value for location selection. For example, if $h_i = h_j$ $(i < j)$, the cloud user needs to re-computes $h_j = \mathsf{H}(key, h_j)$. It means that one location of bit array can be set to 1 only once. Finally, the cloud user can extract the 256-bit values at the corresponding locations of the bit array from C_2 and concatenate the last 256-bit of C_2 as the $gathered$ package. Compared with the above scheme, Bloom filter-based location selection method no longer needs to store 256 hash values but only a bit array (the length of a bit array is the same as the length of one hash value), which reduces the storage overhead. In addition, by using a bit array to record random locations, our location selection method avoids the time to sort 256 hash values, which reduces the computational overhead.

2. The different cloud users always can generate the same file key for the same file in our scheme.

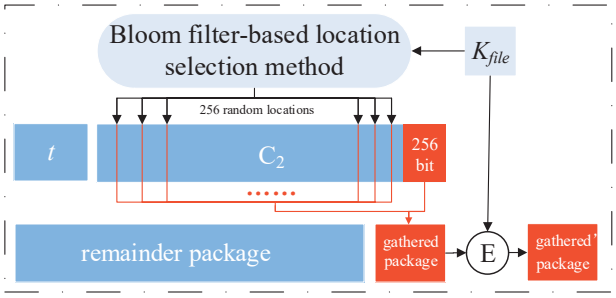


Fig. 5. Package generation of our scheme.

5.3 A Concrete Scheme

The workflow of the $gathered'$ and $remainder$ packages generation is shown in Fig. 5. Our data deduplication scheme involves three operations defined as follows:

Message Upload: If a cloud user wants to upload a file F, the cloud user first splits F into a set of chunks $\{M\}$. For each chunk, the cloud user runs the processes as follows (the procedure of message upload operation is shown in Fig. 6.):

MLE key generation. The cloud user generates the MLE key based on the verifiable OPRF (oblivious pseudorandom function). The RSA-OPRF scheme is based on the RSA blind signature [34]. The detail is described as following:

- The RSA exponent e is fixed and as a public parameter of the scheme. The key server first runs a key generation algorithm with input e to generate N, d such that $ed \equiv 1 \mod \phi(N)$, where modulus N is the product of two distinct primes of roughly equal length. Then, $(N, (N, d))$ is output as the public key, secret key pair.
- A cloud user computes the hash value $h_1 = \mathsf{H}(M)$ of message M and chooses a random number r from the $\mathbb{Z}_N$. Then, the cloud user computes blinded hash

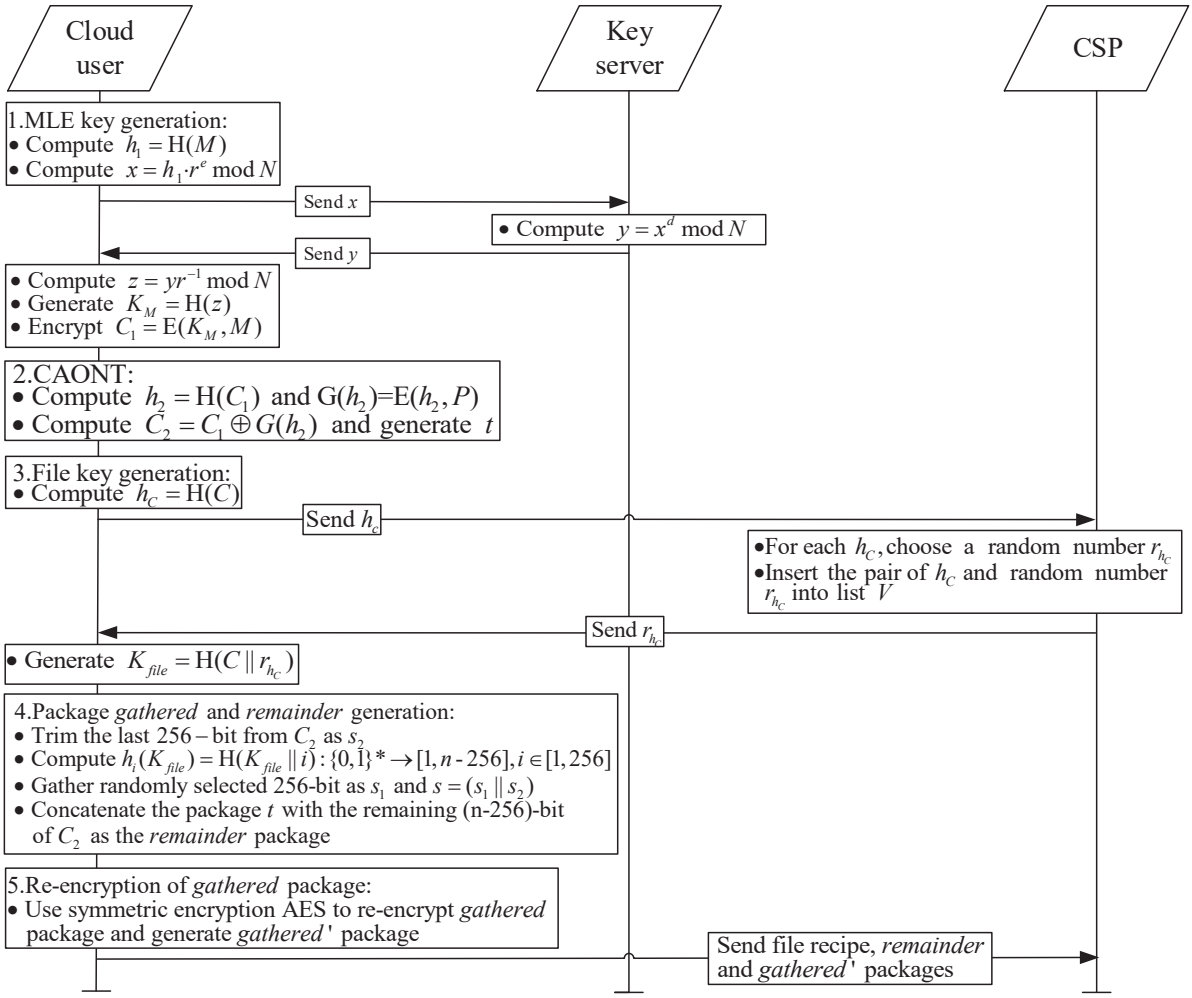


Fig. 6. The procedure of message upload operation.

value $x = h_1 \cdot r^e \mod N$ and sends x to the key server via a secure channel (e.g., SSL/TLS).

- After receiving the blinded hash value x, the key server computes the RSA signature on blinded hash value $y = x^d \mod N$ and returns y to the cloud user.
- Finally, the cloud user computes $z = y \cdot r^{-1} \mod N$ and checks whether $z^e \mod N = h_1$. If $z^e \mod N = h_1$, the cloud user gets the MLE key $K_M = \mathsf{H}(z)$ and uses the K_M to encrypt M and generate the ciphertext $C_1 = \mathsf{E}(K_M, M)$.

CAONT. Base on the CAONT mechanism, the cloud user transforms package C_1 into package t and package C_2. The detail is described as following:

- For the package C_1, a cloud user first computes the hash value $h_2 = \mathsf{H}(C_1)$ as input of pseudo-random mask G and computes $\mathsf{G}(h_2) = \mathsf{E}(h_2, P)$, where $\mathsf{E}(\cdot)$ denotes a symmetric key encryption algorithm (e.g., AES-256) and P denotes a public block with the same size as the message C_1.
- The cloud user computes $C_2 = C_1 \oplus \mathsf{G}(h_2)$.
- Finally, the cloud user divides C_2 into a set of fixed-sized pieces, each with the same size as h_2. Then, the cloud user performs XOR operation of all the pieces as well as h_2 to generate the package t. Without getting the entire message C_2, the result of self-XOR cannot be predicted.

File key generation. We use C to denote the ciphertext of file F. To generate the file key K_{file} of the file F, the workflows are described as following:

- A cloud user sends the hash value $h_C = \mathsf{H}(C)$ to the CSP[3].
- After receiving the hash value h_C, the CSP first checks whether the hash value h_C is in the list V. If the hash value already exists in the list V, the CSP returns the random number r_{h_C} corresponding to the hash value h_C. Otherwise, the CSP randomly chooses a number r_{h_C} for the unique h_C and inserts the pair of hash value h_C and random number r_{h_C} into the list V. Then, the CSP sends the random number r_{h_C} to the cloud user.
- After receiving the random number r_{h_C}, the cloud user generates the file key $K_{file} = \mathsf{H}(C||r_{h_C})$.

Package gathered and remainder generation. To generate the 512-bit *gathered* package $s = (s_1||s_2)$, we use the Bloom

3. In the CSP setup phase, the CSP sets up a random number list V which is used to store the pair of the hash value and random number.

filter-based location selection method in our scheme. The detail is described as following:

- The cloud user trims the last 256-bit from C_2 as s_2.
- The cloud user sets a bit array of $n-256$ bits and 256 independent hash functions defined as follows: $h_i(K_{file}) = \mathsf{H}(K_{file}||i) : \{0,1\}^* \rightarrow [1, n-256]$. These hash functions map each element to a random number uniform over $\{1, ..., n-256\}$. In the initialization stage, all bits of the bit array are set to 0.
- To select 256 random bit array locations, the cloud user uses the file key K_{file} to compute $h_i(K_{file}) = \mathsf{H}(K_{file}||i) : \{0,1\}^* \rightarrow [1, n-256]$. Then, all 256 locations of the bit array are set to 1[4].
- The cloud user extracts the 256-bit values at the corresponding locations of the bit array from C_2 as s_1 and generates the $gathered$ package $s = (s_1||s_2)$.
- Finally, the cloud user concatenates the package t with the remaining $(n-512)$-bit of the ciphertext C_2 as the $remainder$ package.

Re-encryption of gathered package. The cloud user uses the file key K_{file} to re-encrypt the $gathered$ package and generate the $gathered'$ package by using symmetric encryption such as AES-256. The key server encrypts the file key K_{file} by using CP-ABE based on the policy of the file.

As described before, we use the unique identity of the cloud users as the attribute and issue each cloud user a CP-ABE private key, which is related to the identity. The policy of each file as an access tree and the identities of all authorized cloud users are connected with the OR gate. This makes only the authorized cloud user can get the file key. Finally, the $remainder$ and $gathered'$ packages are uploaded to the CSP. The file recipe, which includes the file name, file size and the number of chunks, is also uploaded to the CSP by the cloud user. The file recipes and encrypted file key are used for the other cloud users to recover the original message.

After receiving these packages, the CSP performs deduplication on the $remainder$ and $gathered'$ packages. Once there are some new cloud users to upload the $remainder$ and $gathered'$ packages, the CSP makes a comparison for the stored $remainder$ and $gathered'$ packages and the new $remainder$ and new $gathered'$ packages. If the same $remainder$ and $gathered'$ packages are found, it means the same $remainder$ and $gathered'$ packages have been stored in the CSP. Then, the CSP no longer stores the new packages for saving storage space. The CSP also recalls the key server to add the new cloud users' identity as the attribute in the CP-ABE access tree.

Message Download: If a cloud user wants to download the file F, the cloud user first downloads the file recipe, encrypted file key, and metadata from the CSP. Then, the cloud user downloads all $remainder$ and $gathered'$ packages from the CSP with the help of file receipt and performs the following steps:

Re-decryption of $gathered'$ package. To re-decrypt the $gathered'$ package, the cloud user first obtains the file key K_{file} from the CP-ABE access tree. Then, the cloud user uses K_{file} to decrypt the $gathered'$ package and generate the $gathered$ package $s = (s_1||s_2)$.

Package C_2 and t generation. To recover the package C_2 and t, the following processes need to be performed:

- The cloud user first uses the $remainder$ package to concatenate the s_2 of the $gathered$ package.
- The cloud user sets a bit array of $n-256$ bits (all bits of the bit array are set to 0) and computes $h_i(K_{file}) = \mathsf{H}(K_{file}||i) : \{0,1\}^* \rightarrow [1, n-256]$ $(1 \leq i \leq 256)$. Then, all 256 locations of the bit array are set to 1.
- The cloud user inserts the i-th bit in s_1 into the location of i-th 1 of the bit array in the $remainder$ package from $i = 1$ to $i = 256$.
- Finally, the cloud user generates the package t by trimming the first $|t|$-bit of the $remainder$ package. The remaining part of the $remainder$ package is precisely the ciphertext C_2.

The reversion of CAONT. To reconstruct the package C_1, the processes are described as following:

- The cloud user first divides C_2 into a set of fixed-sized pieces, each with the same size as t.
- The cloud user performs XOR operation of all the pieces as well as t to generate h_2 and compute $C_1 = C_2 \oplus \mathsf{G}(h_2)$.

Finally, the cloud user checks whether $h(C_1) = h_2$. If $h(C_1) = h_2$, the cloud user uses the MLE key K_M to decrypt C_1 and get the plaintext $M = \mathsf{D}(K_M, C_1)$. The cloud user integrates all the set of chunks $\{M\}$ into file F, with the help of file receipt.

Dynamic access control: If the file key is compromised or expired, the cloud user should re-encrypt the stored file for protecting the confidentiality of the file. To re-encrypt the file, the cloud user (on behalf of the owner of file F) first downloads the file recipe, encrypted file key, and metadata from the CSP. The cloud user also downloads all the $remainder$ and $gathered'$ packages from the CSP with the help of file receipt. Then, the cloud user performs the following steps:

Re-decryption of $gathered'$ package. To re-decrypt the $gathered'$ package, the cloud user first obtains the file key K_{file} from the CP-ABE access tree. Then, the data owner uses K_{file} to decrypt the $gathered'$ package and generate the $gathered$ package $s = (s_1||s_2)$.

Package C_2 and t generation. To recover the package C_2 and t, the following processes need to be performed:

- To recover the package C_2, the cloud user first uses the $remainder$ package to concatenate the s_2 of the $gathered$ package.
- The cloud user sets a bit array of $n-256$ bits (all bits of the bit array are set to 0) and computes $h_i(K_{file}) = \mathsf{H}(K_{file}||i) : \{0,1\}^* \rightarrow [1, n-256]$ $(1 \leq i \leq 256)$. Then, all 256 locations of the bit array are set to 1.
- The cloud user inserts the i-th bit in s_1 into the location of i-th 1 of the bit array in the $remainder$ package from $i = 1$ to $i = 256$.
- Finally, the cloud user generates the package t by trimming the first $|t|$-bit of the $remainder$ package.

4. For a file F, the Bloom filter-based location selection method only needs to be run once.

The remaining part of the $remainder$ package is precisely the ciphertext C_2.

File key updating. To generate the new file key K'_{file} for the file F, the cloud user performs the following steps:

- The cloud user first sends the hash value h_C and random number r_{h_C} to the CSP.
- After receiving the hash value h_C and random number r_{h_C}, the CSP chooses a new random number r'_{h_C} for the unique h_C. Then the CSP sends the new random number r'_{h_C} to the cloud user.
- After receiving the random number r'_{h_C}, the cloud user generates the new file key $K'_{file} = \mathsf{H}(C||r'_{h_C})$.

New-gathered and new-remainder packages generation. To generate the 512-bit new-$gathered$ package $s' = (s'_1||s'_2)$ and new-$remainder$ package, we use the Bloom filter-based location selection method in our scheme. The detail is described as following:

- To generate the 512-bit new-$gathered$ package s', the cloud user first trims the last 256-bit from C_2 as s'_2.
- The cloud user sets a bit array of $n-256$ bits and all bits of the bit array are set to 0.
- The cloud user uses the new file key K'_{file} and the Bloom filter-based location selection method to generate the random locations of C_2 by computing $h_i(K'_{file}) = \mathsf{H}(K'_{file}||i) : \{0,1\}^* \rightarrow [1, n-256], i \in [1, 256]$. Then, all 256 locations of the bit array are set to 1.
- Finally, the cloud user extracts the 256-bit values at the corresponding locations of the bit array from C_2 as s'_1 and generates the new-$gathered$ package $s' = (s'_1||s'_2)$. The remaining package t and the large part of the ciphertext C_2 are also gathered as the new-$remainder$ package.

Re-encryption of new-gathered package. The cloud user uses the new file key K'_{file} to re-encrypt the new-$gathered$ package and generate the new-$gathered'$ package by using symmetric encryption such as AES-256. The new file key is encrypted based on the new policy of the file and is distributed by using the CP-ABE mechanism.

Finally, the cloud user uploads the new-$remainder$ and new-$gathered'$ packages to the CSP.

Remark 2. *It should be emphasized that only when the file key is expired or cloud users are revoked, the data owner needs to re-encrypt these packages. However, if some $remainder$ and $gathered$ packages are already stored in the CSP and a new cloud user wants to upload the same packages, these packages do not require re-encryption. The CP-ABE key server only needs to add the identity of the new cloud user into the group. In addition, our scheme can implement lazy revocation. In lazy revocation, re-encryption of stored packages are deferred for a period of time (one day or one week). The data owner can avoid staying online.*

6 Analysis of Our Proposed Scheme

6.1 Security Analysis

In this section, we analyze the security of our scheme involving three parts: integrity, confidentiality and resistance to stub-reserved attack. We assume that the underlying basic tools are secure, which include the message-locked encryption scheme, symmetric encryption scheme, convergent all-or-nothing transform scheme, Bloom filter scheme, and CP-ABE scheme. The security of our scheme is ensured by these assumptions.

Theorem 6.1. *The proposed scheme guarantees data integrity.*

Proof. In the cloud environment, an attacker will tamper or attack the sensitive outsourced data. The data integrity of cloud users' sensitive data may be corrupted. In our scheme, the data integrity can be checked by computing the hash of package C_1 and checking whether it equals h_2.

After downloading the $gathered'$ and $remainder$ packages from the CSP, the cloud user first re-decrypts the $gathered'$ package and generates the $gathered$ package. Then, the cloud user recovers the package C_2 and t by using the Bloom filter-based location selection method. Secondly, the cloud user divides C_2 into a set of fixed-sized pieces, each with the same size as t. Then, the cloud user performs XOR operation of all the pieces as well as t to generate h_2 and compute $C_1 = C_2 \oplus \mathsf{G}(h_2)$. Finally, the cloud user checks whether $\mathsf{H}(C_1) = h_2$. Because the cloud user generates the package h by computing self-XOR operation, it is possible that the self-XOR operation returns a correct h even if the package has been tampered. Specifically, a smart adversary can divide C_2 into fixed-size pieces and flips the same bit location for an even number of the pieces. However, a corrupted package will be reverted to an incorrect C'_1 and $\mathsf{H}(C'_1) \neq h$. Then, the cloud user can detect that message integrity was destroyed. Therefore, our scheme guarantees data integrity. □

Theorem 6.2. *The proposed scheme guarantees data confidentiality.*

Proof. In our scheme, the procedure of data re-encryption is as follows. A cloud user first uses the DupLESS scheme to encrypt the message M and generate the ciphertext C_1. Then, the cloud user uses CAONT scheme to transform ciphertext C_1 into the $gathered$ and $remainder$ packages. Finally, the cloud user uses symmetric encryption (such as AES-256) to re-encrypt the $gathered$ package and get the $gathered'$ package. To prove the data confidentiality, we need to prove the confidentiality of data in the procedure of $gathered$ package re-encryption, CAONT transformation and MLE encryption.

Firstly, we analyze the confidentiality of $gathered$ package re-encryption. In our scheme, the cloud user uses symmetric encryption (such as AES-256) to re-encrypt the $gathered$ package. The key is protected and distributed by using CP-ABE. Since the CP-ABE access tree does not include the identities of adversary, unauthorized cloud users and revoked cloud users, the adversary cannot access the key even if he colludes with the unauthorized and revoked cloud users. We assume that AES and CP-ABE are secure, the confidentiality of $gathered$ package can be guaranteed.

Secondly, we need to prove that if the adversary cannot get the $gathered$ package, the probability that the adversary can recover the C_1 package is negligible. As described in [28], CAONT inherits the security properties of the original AONT [29]. According to the theorem 1 in [29]: suppose $l \leq k_0$ and $k_0 \geq 14$. Suppose that there

TABLE 1 Comparison of Data Deduplication Schemes

Scheme	Encrypted data deduplication	Data integrity	User joining	User revocation	Lightweight re-encryption
CE [17]	Yes	No	No	No	No
Hur *et al.* [21]	Yes	Yes	Yes	Yes	No
REED [25]	Yes	Yes	Yes	No	Yes
Our scheme	Yes	Yes	Yes	Yes	Yes

TABLE 2 Computational Cost of Re-encryption

Chunk Size	Traditional re-encryption		Our Scheme	
	Re-encryption	Re-decryption	Re-encryption	Re-decryption
4 KB	$100\%Enc$	$100\%Dec$	$Gen + 1.56\%Enc$	$Rec + 1.56\%Dec$
8 KB	$100\%Enc$	$100\%Dec$	$Gen + 0.78\%Enc$	$Rec + 0.78\%Dec$
16 KB	$100\%Enc$	$100\%Dec$	$Gen + 0.39\%Enc$	$Rec + 0.39\%Dec$

exists an adversary A that $(T, q_G, q_H, \varepsilon) - distinguishes$ OAEP[5] with l missing bits, where $q_G \leq 2^{k_0-1}$. Then $\varepsilon \leq 8q_G\frac{k_0}{log_2 k_0}2^{-l}$. We can conclude that the adversary's advantage in $(T, q_G, q_H, \varepsilon) - distinguishes$ CAONT-OAEP is less than or equal to $8q_G\frac{256}{log_2 256}2^{-256}$ (where $l = 256$ and $k_0 = 256$ in our scheme). The probability of an adversary to recover the package C_1 without the $gathered$ package is less than or equal to $\frac{1}{2^{248}}q_G$. Thus, if the adversary cannot get the $gathered$ package, the probability that the adversary can recover package C_1 is negligible.

According to the above proof, since the adversary cannot compromise the file key, all $gathered$ and $remainder$ packages cannot be reverted. Thus, our scheme achieves the same level of confidentiality as DupLESS [35]. In detail, if the key server is secure, then the ciphertexts appear to be derived from a random key-space. Thus, our scheme guarantees confidentiality even for the predictable data. Even if the key server is compromised, the confidentiality of unpredictable data is still preserved. □

Resistance to Stub-reserved Attack: Once a cloud user is revoked from the group, the data owner re-encrypts the packages as follows: 1) The data owner first re-decrypts the $gathered'$ package and gets the $gathered$ package. Then, the data owner uses the $gathered$ and $trimmed$ packages to recover the package t and C_2. 2) The data owner chooses a new file key K'_{file}. Then, the data owner uses the new file key K'_{file} to generate the random 256 locations of C_2. 3) The data owner gathers the newly selected 256-bit and the last 256-bit of C_2 to generate a new-$gathered$ package, the package t and the remained a large part of C_2 are gathered as a new-$remainder$ package. 4) The data owner re-encrypts the new-$gathered$ package and generates a new-$gathered'$ package. Because the new-$gathered$ package is selected by randomly choosing from C_2 package, the revoked cloud user cannot predict which part of the package will be re-encrypted. The revoked cloud user needs to preserve all $gathered$ and $remainder$ packages to recover the original message. In this case, it is better to keep the original message. Thus, our scheme can prevent the stub-reserved attack.

5. The random oracle $\mathcal{O}$ could be used to simulate random oracles G and H, with the condition that at most q_G queries are made to random oracle G and at most q_H queries are made to random oracle H.

6.2 Comparison

Table 1 presents the comparison among four data deduplication schemes, which consist of the convergent encryption (CE) scheme [17], secure data deduplication scheme with dynamic ownership management [21], REED scheme [25] and our scheme, in terms of encrypted data deduplication, data integrity, user joining, user revocation, and lightweight re-encryption.

All the data deduplication schemes support data deduplication on ciphertext, which can prevent unauthorized cloud users and the CSP from accessing the plaintext and guarantee data privacy. CE scheme cannot guarantee data integrity. The cloud users' data is vulnerable to the poison attack. However, by employing an additional mechanism, other data deduplication schemes enable the cloud users to check the integrity of their sensitive data.

CE scheme neglects the problem of dynamic ownership management and cannot support user joining and user revocation. In Hur *et al.'s* scheme, for the universe of cloud users who own the same data, the cloud server sets an ownership group. The cloud server can use the binary KEK (key-encrypting key) tree for distributing the group key. Therefore, Hur *et al.'s* scheme can support user joining and user revocation. By using the CAONT mechanism, the REED scheme can achieve lightweight re-encryption. However, the REED scheme is vulnerable to the stub-reserved attack and cannot support user revocation.

Based on the CAONT and Bloom filter-based location selection method, we propose a new data deduplication with efficient re-encryption. Once a cloud user is revoked or file key is compromised, the data owner re-encrypts sensitive data by using the Bloom filter-based location selection method and CAONT mechanism. Due to the property of one-way hash function, our scheme can resist the stub-reserved attack and guarantee the privacy of cloud users' data. Besides, instead of re-encrypting the entire package, data owners are only required to re-encrypt a small part of it through the CAONT. Thus, our scheme can support user joining, user revocation, and lightweight re-encryption.

Table 2 presents the computational cost of re-encryption among the traditional schemes [21], [36] and our scheme. Enc and Dec denote the cost of encryption and decryption, respectively. Gen and Rec denote the cost of $gathered$ package generation and recovering, respectively.

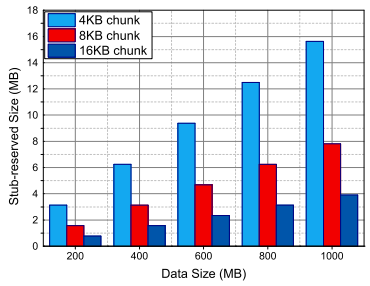


Fig. 7. Stub-reserved size.

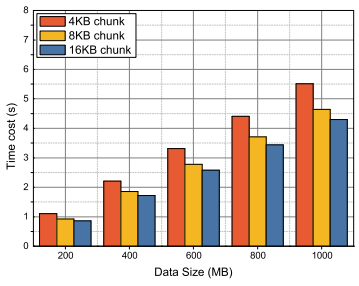


Fig. 8. Recovering time.

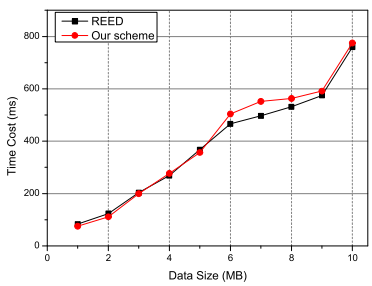


Fig. 9. MLE key generation time.

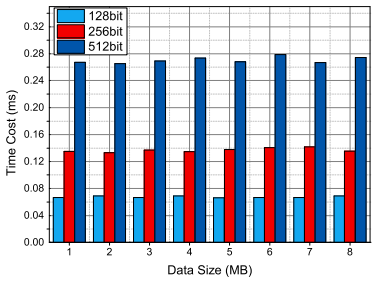


Fig. 10. Location generation time.

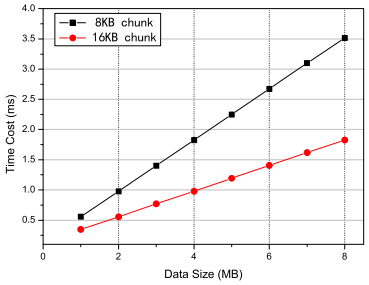


Fig. 11. *gathered* package generation time.

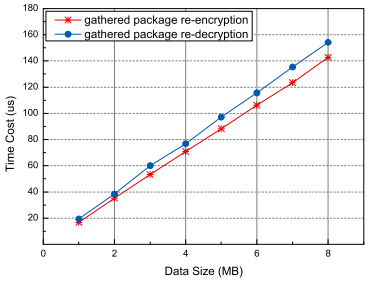


Fig. 12. *gathered* package re-encryption and re-decryption time.

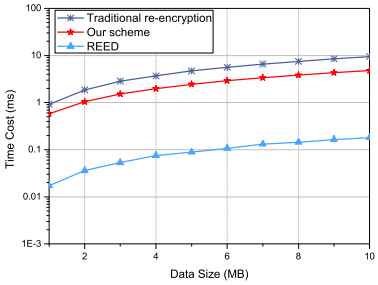


Fig. 13. Computation time for re-encryption.

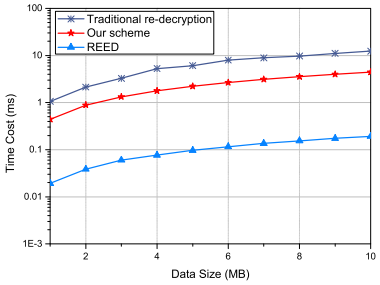


Fig. 14. Computation time for re-decryption.

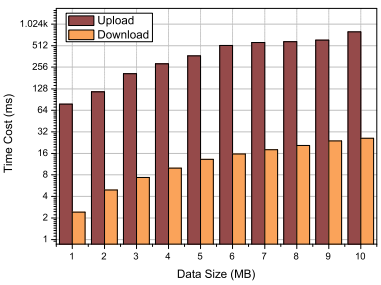


Fig. 15. Computation time for upload and download.

7 Performance Evaluation

In this section, we provide a thorough experimental evaluation of our scheme. We implement our scheme in the Java programming language by employing java.security and javax.crypto packages. The testing environment is Intel(R) Xeon(R) E5-1620 v3 3.50GHz CPU 16.0GB RAM, Ubuntu v14.04. We use both of the synthetic datasets and real-world public dataset collected by the File systems and Storage Lab (FSL) at Stony Brook University [37]. The FSLhomes dataset contains snapshots of students' home directories, where files consist of source code, binaries and office documents. We focus on the FSLhomes dataset in 2014, which comprises 174 daily snapshots from January 18 to November 26, 2014. To verify the validity of the stub-reserved attack and compare the performance of our scheme with the existing deduplication schemes [21], [25], [26], we mainly use stub-reserved size, plaintext recovering time, MLE key generation time, re-encryption and re-decryption time as the evaluation metrics. Stub-reserved size is the size of the data that the revoked cloud user needs to preserve for recovering the plaintext message when executing the stub-reserved attack. Plaintext recovering time consists of re-decryption, CAONT transformation and MLE decryption time. MLE key generation time is the time required by the cloud user and key server to generate the MLE key. Re-encryption and re-decryption time are the time required for cloud users to re-encrypt and re-decrypt data. We observe the impacts of varying chunk size and data size on the scheme performance. According to the paper [25], [26], [38], we set the chunk size to 4 KB, 8 KB and 16 KB (larger sizes have performance that degrades as expected due to inputs not fitting into the CPU cache). The data size ranges from 1 MB to 10 MB [21]. Our results are averaged over 20 runs.

Stub-reserved attack performance: We first measure the performance of stub-reserved attack proposed in this paper. In stub-reserved attack, the revoked cloud user can recover the original plaintext by reserving a small part of

the package. We test the size of reserved $stub$ package for recovering the original plaintext, the data size ranges from 200 MB to 1000 MB and the average chunk sizes at 4 KB, 8 KB, and 16 KB [25], [26]. The data size of the $stub$ package that needs to be reserved is shown in Fig. 7.

As shown in the experimental results, to recover the plaintext, a revoked cloud user needs to preserve 1.56% of the entire package for 4 KB chunk, 0.78% of the entire package for 8 KB chunk and 0.39% of the entire package for 16 KB chunk. In particular, a revoked cloud user only needs to preserve 3.90625 MB data for recovering the size of 1000 MB data (for chunk sizes at 16 KB). In addition, we measure the recovering time and the results shown in Fig. 8.

MLE key generation performance: We measure the performance in MLE key generation. To reduce the round-trip overhead of small requests, we batch multiple MLE key generation requests. We test the MLE key generation time of REED and our scheme from 1 MB to 10 MB. According to the paper [25], [26], we set the batch size to 256 and chunk size to 8 KB. The cost computation time for MLE key generation is shown in Fig. 9. The results show that the MLE key generation time in our scheme is almost the same as the REED scheme.

Location generation performance: In this paper, we proposed the Bloom filter-based location selection method to choose random locations. Although we choose 256-bit random locations as s_1 in our scheme. To achieve more efficiency or security, a cloud user also can select 128-bit or 512-bit random locations as s_1. We measure the location selection time of different bit lengths and the result is shown in Fig. 10.

Gathered package generation performance: We measure the performance of $gathered$ package generation. The $gathered$ package generation time consists of location generation, bits extraction and concatenation time. According to the paper [25], [26], we set the chunk size to 8 KB and 16 KB. The cost computation time for $gathered$ package generation is shown in Fig. 11. The results show that the overhead of $gathered$ package generation for 16 KB chunk is less than 8 KB chunk. This is because the large size of chunk requires less processing overhead.

For protecting the data confidentiality, once the file key is compromised or expired, a cloud user needs to re-encrypt the $gathered$ package. We measure the $gathered$ package re-encryption and re-decryption time, in which we set the chunk size to 8 KB. The results are shown in Fig. 12.

Re-encryption and re-decryption performance: In traditional re-encryption scheme [21], [36], a cloud user needs to re-encrypt the entire file, which costs abundant computing resources. To achieve efficient re-encryption, we propose a secure and efficient re-encryption method for data deduplication based on the CAONT and Bloom filter-based location selection method. We measure the re-encryption and re-decryption performance of our scheme and the existing schemes [21], [25], [26]. We set the chunk size to 8 KB and the data size from 1 MB to 10 MB. The results are depicted in Fig. 13. and Fig. 14. Although our scheme is slower than the REED scheme, our scheme can prevent the stub-reserved attack and still more efficient than the traditional re-encryption scheme.

Upload and download performance: We measure the upload and download performance of our scheme. To test the computation time of upload and download, we choose the message size from 1 MB to 10 MB and set the chunk size to 8 KB. The upload time includes MLE key generation, message-locked encryption, CAONT, package generation and re-encryption time. The download time includes package re-decryption, package recovering, CAONT and message-locked decryption time. The evaluation result is shown in Fig. 15. The result indicates that the upload time is mainly bounded by the MLE key generation.

In summary, our scheme has the same MLE key generation time as the REED scheme [25], [26]. The MLE key generation speed of our scheme can reach 12 MB/s. Compared with the traditional re-encryption scheme [21], [36], our scheme only costs 63.2% and 41.5% of the computing time in data re-encryption and re-decryption, respectively. The re-encryption speed and re-decryption speed of our scheme can reach 1.71 MB/ms and 2.26 MB/ms (the traditional re-encryption speed and re-decryption speed are 1.08 MB/ms and 0.94 MB/ms). Thus, our data deduplication scheme is efficient in re-encryption and re-decryption.

8 Related Work

Numerous researchers have devoted considerable attention to the problem of how to support data deduplication under ciphertext. CE scheme is the first clever solution for data deduplication over encrypted data [17]. The main idea is as follows: a user encrypts and decrypts some sensitive data with a convergent key, which is derived by hashing these data. Since the convergent key and data are deterministic, the identical data is deterministically encrypted to the same ciphertext, no matter who encrypts them. Thus, this allows cloud service providers to perform deduplication over ciphertexts. However, the CE scheme is inherently vulnerable to the brute-force dictionary attacks. In order to solve this problem, Bellare *et al*. [35] proposed the DupLESS scheme, in which a user obtains the key from a dedicated key-server via an oblivious PRF (OPRF) protocol. The OPRF mechanism is used to "blind" the fingerprint. Based on the RSA mechanism, the key server is configured with a system-wide public/private key pair. This enables the key server to return the MLE key without knowing the original fingerprint. The rate-limits are used in the key generation requests and can be efficient against the brute-force attacks. If the key-server is secure, the encryption key appears to be derived from a random space. Shin *et al*. [39] extended the predicate encryption scheme in data deduplication. However, this scheme only supports the requirement of single-user data deduplication. By introducing the additional tag checking mechanism, Bellare *et al*. proposed the randomized convergent encryption (RCE) scheme [38]. After decrypting the ciphertext, the user uses the plaintext to generate the tag and compare it with the corresponding tag. If tags are consistency, the user accepts the ciphertext; else, rejects it. Thus, the RCE scheme guarantees the integrity of users' data. However, these schemes suffer from security flaws with respect to user revocation. If the revoked users keep the convergent key, they can access the plaintext without permission. Thus, the confidentiality of users' sensitive data cannot be guaranteed.

To deal with the problem of efficient and reliable key management, Li *et al*. [40] proposed a secure data deduplication scheme by employing a security Ramp secret sharing scheme [41]. To realize dynamic updates in the deduplication, Wen *et al*. [36] proposed a session-key-based convergent key management scheme and convergent key sharing scheme. To deal with the problem of dynamic ownership changes of outsourced data, Hur *et al*. [21] used the group key to re-encrypt the ciphertext, which allows only the authorized cloud user to access the shared data. Chen *et al*. [42] proposed a block-level message-locked encryption (BL-MLE) scheme, which achieves file-level and block-level deduplication. To flexibly support data access control and revocation, Yan *et al*. [43] proposed a scheme to deduplicate encrypted data stored in the cloud. Based on static or dynamic decision trees, Jiang *et al*. introduced a new primitive called R-MLE2 and proposed a cloud data deduplication with randomized tag [23]. Based on the PAKE protocol, Liu *et al*. [44] proposed a secure data deduplication scheme, which supports the client-side encryption without an additional independent server. To address the problem of authorized data deduplication, Li *et al*. proposed several data deduplication schemes supporting authorized duplicate checking in a hybrid cloud architecture [45]. However, these schemes mainly use the method of re-encryption to solve the problem of user revocation. The traditional re-encryption scheme inevitably brings abundant computation overhead.

Recently, efficient re-encryption techniques are attracting widespread attention in the scientific community. To achieve efficient re-encryption and lightweight rekeying in data deduplication, Li *et al*. [25] proposed a rekeying-aware encrypted deduplication storage system. In this scheme, a data owner does not need to re-encrypt the entire package but only a small part of it, saving excessive computation overhead. In addition, the authors extended REED with ciphertext-policy attribute-based encryption [30] to control the access privileges to different data. However, a security weakness is found in the REED scheme. That is, the REED scheme is vulnerable to the stub-reserved attack. The detailed analysis of this scheme is given in Section 3.

9 Conclusion

In this paper, we propose a Bloom filter-based location selection method and a secure data deduplication scheme with efficient re-encryption. Owing to the inherent property of one-way hash function, our scheme is secure against the stub-reserved attack and guarantees the data privacy of the data owners' sensitive data. In addition, instead of re-encrypting the entire package, data owners are only required to re-encrypt a small part of it through the CAONT, which saves excessive computation overhead. We also prove that our scheme can achieve the desired security goals and provide detailed simulation tests. The experimental results show that our scheme is efficient in re-encryption.

Acknowledgments

This work is supported by the National Natural Science Foundation of China (Nos. 61702401, 61702402, 61960206014), China 111 Project (Grant No. B16037) and Xidian University (No.20109194858).

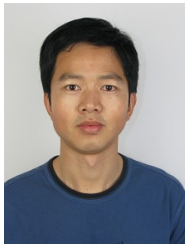
References

[1] X. Chen, J. Li, J. Weng, J. Ma, and W. Lou, "Verifiable computation over large database with incremental updates," *IEEE Trans. Computers*, vol. 65, no. 10, pp. 3184–3195, 2016.
[2] M. Gerla, J. Weng, and G. Pau, "Pics-on-wheels: Photo surveillance in the vehicular cloud," *International Conference on Computing, Networking and Communications*, pp. 1123–1127, 2013.
[3] X. Chen, J. Li, J. Ma, Q. Tang, and W. Lou, "New algorithms for secure outsourcing of modular exponentiations," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 9, pp. 2386–2396, 2014.
[4] H. Yuan, X. Chen, T. Jiang, X. Zhang, Z. Yan, and Y. Xiang, "Dedupdum: Secure and scalable data deduplication with dynamic user management," *Inf. Sci.*, vol. 456, pp. 159–173, 2018.
[5] H. Huang, X. Chen, Q. Wu, X. Huang, and J. Shen, "Bitcoin-based fair payments for outsourcing computations of fog devices," *Future Generation Comp. Syst.*, vol. 78, pp. 850–858, 2018.
[6] IDC. (2014) The digital universe of opportunities : Rich data and the increasing value of the internet of things. [Online]. Available: https://www.emc.com/leadership/digital-universe/2014iview/index.htm
[7] W. J. Bolosky, S. Corbin, D. Goebel, and J. R. Douceur, "Single instance storage in windows 2000," in *Conference on Usenix Windows Systems Symposium*, 2000.
[8] Dropbox. (2007). [Online]. Available: http://www.dropbox.com
[9] GoogleDrive. (2012). [Online]. Available: http://drive.google.com
[10] Memopal. (2018). [Online]. Available: http://www.memopal.com
[11] Netapp. (2008) Netapp deduplication helps duke institute for genome sciences and policy reduce storage requirements for genomic information by 83 percent. [Online]. Available: http://www.netapp.com
[12] M. Dutch, "Understanding data deduplication ratios," in *SNIA Data Management Forum*, 2008, pp. 1–13.
[13] T. Jiang, X. Chen, J. Li, D. S. Wong, J. Ma, and J. K. Liu, "TIMER: secure and reliable cloud storage against data re-outsourcing," *Information Security Practice and Experience - 10th International Conference*, pp. 346–358, 2014.
[14] X. Chen, B. Lee, and K. Kim, "Receipt-free electronic auction schemes using homomorphic encryption," *Information Security and Cryptology - ICISC 2003, 6th International Conference, Seoul, Korea, November 27-28, 2003, Revised Papers*, pp. 259–273, 2003.
[15] J. Wang, X. Chen, J. Li, K. Kluczniak, and M. Kutylowski, "Trdup: enhancing secure data deduplication with user traceability in cloud computing," *IJWGS*, vol. 13, no. 3, pp. 270–289, 2017.
[16] X. Zhang, X. Chen, J. Wang, Z. Zhan, and J. Li, "Verifiable privacy-preserving single-layer perceptron training scheme in cloud computing," *Soft Comput.*, vol. 22, no. 23, pp. 7719–7732, 2018.
[17] J. R. Douceur, A. Adya, W. J. Bolosky, P. Simon, and M. Theimer, "Reclaiming space from duplicate files in a serverless distributed file system," in *ICDCS*, 2002, pp. 617–624.
[18] L. D. Stein, "The case for cloud computing in genome informatics," *Genome Biology*, vol. 11, no. 5, pp. 207–207, 2010.
[19] GoogleGenomics. (2018). [Online]. Available: https://cloud.google.com/genomics/
[20] Amazon. (2018). [Online]. Available: https://aws.amazon.com/
[21] J. Hur, D. Koo, Y. Shin, and K. Kang, "Secure data deduplication with dynamic ownership management in cloud storage," *IEEE Trans. Knowl. Data Eng.*, vol. 28, no. 11, pp. 3113–3125, 2016.
[22] J. Li, X. Chen, X. Huang, S. Tang, Y. Xiang, M. M. Hassan, and A. Alelaiwi, "Secure distributed deduplication systems with improved reliability," *IEEE Trans. Computers*, vol. 64, no. 12, pp. 3569–3579, 2015.
[23] T. Jiang, X. Chen, Q. Wu, J. Ma, W. Susilo, and W. Lou, "Secure and efficient cloud data deduplication with randomized tag," *IEEE Trans. Information Forensics and Security*, vol. 12, no. 3, pp. 532–543, 2017.
[24] W. C. G. III, A. Shull, S. Myers, and A. J. Lee, "On the practicality of cryptographically enforcing dynamic access control policies in the cloud," *IEEE Symposium on Security and Privacy*, pp. 819–838, 2016.
[25] J. Li, C. Qin, P. P. C. Lee, and J. Li, "Rekeying for encrypted deduplication storage," in *46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, 2016, pp. 618–629.
[26] C. Qin, J. Li, and P. P. C. Lee, "The design and implementation of a rekeying-aware encrypted deduplication storage system," *TOS*, vol. 13, no. 1, pp. 9:1–9:30, 2017.

[27] R. L. Rivest, "All-or-nothing encryption and the package transform," in *Fast Software Encryption, 4th International Workshop*, 1997, pp. 210–218.
[28] M. Li, C. Qin, and P. P. C. Lee, "Cdstore: Toward reliable, secure, and cost-efficient cloud storage via convergent dispersal," in *2015 USENIX Annual Technical Conference, USENIX ATC '15*, 2015, pp. 111–124.
[29] V. Boyko, "On the security properties of OAEP as an all-or-nothing transform," *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, pp. 503–518, 1999.
[30] J. Bethencourt, A. Sahai, and B. Waters, "Ciphertext-policy attribute-based encryption," in *IEEE Symposium on Security and Privacy*, 2007, pp. 321–334.
[31] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Commun. ACM*, vol. 13, no. 7, pp. 422–426, 1970.
[32] M. van Dijk, A. Juels, A. Oprea, R. L. Rivest, E. Stefanov, and N. Triandopoulos, "Hourglass schemes: how to prove that cloud files are encrypted," *the ACM Conference on Computer and Communications Security, CCS'12*, pp. 265–280, 2012.
[33] D. Harnik, B. Pinkas, and A. Shulman-Peleg, "Side channels in cloud services: Deduplication in cloud storage," *IEEE Security & Privacy*, vol. 8, no. 6, pp. 40–47, 2010.
[34] D. Chaum, "Blind signatures for untraceable payments," in *Advances in Cryptology: Proceedings of CRYPTO '82, Santa Barbara, California, USA, August 23-25, 1982.*, 1982, pp. 199–203.
[35] M. Bellare, S. Keelveedhi, and T. Ristenpart, "Dupless: Server-aided encryption for deduplicated storage," *IACR Cryptology ePrint Archive*, 2013.
[36] M. Wen, K. Ota, H. Li, J. Lei, C. Gu, and Z. Su, "Secure data deduplication with reliable key management for dynamic updates in cpss," *IEEE Trans. Comput. Social Systems*, vol. 2, no. 4, pp. 137–147, 2015.
[37] Fsl traces and snapshots public archive. [Online]. Available: http://tracer.filesystems.org/traces/fslhomes/2014/
[38] B. Mihir, K. Sriram, and R. Thomas, "Message-locked encryption and secure deduplication," *Advances in Cryptology - EUROCRYPT 2013, 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques,*, vol. 28, no. 11, pp. 296–312, 2013.
[39] Y. Shin and K. Kim, "Equality predicate encryption for secure data deduplication," in *Proc. Conf. Inf. Security Cryptol.*, 2012, pp. 64–70.
[40] J. Li, X. Chen, M. Li, J. Li, P. P. Lee, and W. Lou, "Secure deduplication with efficient and reliable convergent key management," *IEEE trans. Parallel Distrib. Syst.*, vol. 25, no. 6, pp. 1615–1625, 2014.
[41] G. R. Blakley and C. Meadows, "Security of ramp schemes," in *Proc. Adv. CRYPTO, Lecture Notes in Computer Science*, vol. 196, 1985, pp. 242–268.
[42] R. Chen, Y. Mu, G. Yang, and F. Guo, "Bl-mle: Block-level message-locked encryption for secure large file deduplication," *IEEE Trans. Inf. Forensics Security*, vol. 10, no. 12, pp. 2643–2652, 2015.
[43] Z. Yan, W. Ding, X. Yu, H. Zhu, and R. H. Deng, "Deduplication on encrypted big data in cloud," *IEEE Trans. Big Data*, vol. 2, no. 2, pp. 138–150, 2016.
[44] J. Liu, N. Asokan, and B. Pinkas, "Secure deduplication of encrypted data without additional independent servers," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, 2015, pp. 874–885.
[45] J. Li, Y. K. Li, X. Chen, P. P. C. Lee, and W. Lou, "A hybrid cloud approach for secure authorized deduplication," *IEEE Trans. Parallel Distrib. Syst.*, vol. 26, no. 5, pp. 1206–1216, 2015.

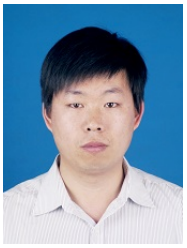
Haoran Yuan received the B.S. degree of network engineering, Xi'an University of Posts & Telecommunications, in 2015. He is currently pursuing the Ph.D. degree with the School of Cyber Engineering, Xidian University. His research interests include cloud security and data security, data deduplication, data auditing and cloud computing security.

Xiaofeng Chen received his B.S. (1998) and M.S. (2000) on Mathematics from Northwest University. He got his Ph.D degree in Cryptography from Xidian University in 2003. Currently, he works at Xidian University as a professor. His research interests include applied cryptography and cloud computing security. He has published over 200 research papers in international conferences and journals. His work has been cited more than 7000 times at Google Scholar. He is in the Editorial Board of IEEE Transactions on Dependable and Secure Computing and Security and Privacy, and Computing and Informatics etc. He has served as the program/general chair or program committee member in over 30 international conferences.



Jin Li received his B.S. (2002) and M.S. (2004) from Southwest University and Sun Yat-sen University, both in Mathematics. He got his Ph.D degree in information security from Sun Yat-sen University at 2007. Currently, he is a professor at Guangzhou University. His research interests include cloud computing security and cryptographic protocols. He has published more than 100 papers in international conferences and journals including IEEE INFOCOM, IEEE Transaction on Parallel and Distributed Computation etc. He has served as the TPC committee for many international conferences.

Tao Jiang received the B.S. degree from Shandong Jianzhu University in 2009, the M.S. degree from Jiangsu University in 2012, and the Ph.D. degree from Xidian University in 2016. He is currently a postdoctoral lecturer at the School of Cyber Engineering in Xidian University. His research interests include applied cryptography and cloud computing security.



Jianfeng Wang received his M.S. degree in Mathematics from Xidian University, China. He got his Ph.D degree in Cryptography from Xidian University in 2016. Currently, he works at Xidian University. He visited Swinburne University of Technology, Australia, from December 2017 to December 2018. His research interests include applied cryptography, cloud security and searchable encryption.

Robert H. Deng has been a professor at the School of Information Systems, Singapore Management University since 2004. His research interests include data security and privacy, multimedia security, network and system security. He was an associate editor of the *IEEE Transactions on Information Forensics and Security* from 2009 to 2012. He is currently an associate editor of the *IEEE Transactions on Dependable and Secure Computing* and *Security and Communication Networks* (John Wiley). He is the cochair of the Steering Committee of the ACM Symposium on Information, Computer and Communications Security. He is a fellow of the IEEE.