Secure Deduplication Storage Systems Supporting Keyword Search

Jin Li ^{a,*}

^{a,1} *School of Computer Science and Educational Software
Guangzhou University, Guangzhou, 510006 P.R. China*

² *State Key Laboratory of Integrated Service Networks (ISN), Xidian University,
Xi'an 710071 P.R. China*

Xiaofeng Chen ^b

^b *State Key Laboratory of Integrated Service Networks (ISN)
Xidian University, Xi'an, P.R. China*

Fatos Xhafa ^c

^c *Department of Languages and Informatics Systems
Technical University of Catalonia, Spain*

Leonard Barolli ^d

^d *Department of Information and Communication Engineering
Fukuoka Institute of Technology, Japan*

Abstract

Data deduplication is an attractive technology to reduce storage space for increasing vast amount of duplicated and redundant data. In a cloud storage system with data deduplication, duplicate copies of data will be eliminated and only one copy will be kept in the storage. To protect the confidentiality of sensitive data while supporting deduplication, the convergent encryption technique has been proposed to encrypt the data before outsourcing. However, the issue of keyword search over encrypted data in deduplication storage system has to be addressed for efficient data utilization. This paper firstly proposes two constructions which support secure keyword search in this scenario. In these constructions, the integrity of the data can be realized by just checking the convergent key, without other traditional integrity auditing mechanisms. Then, two extensions are presented to support fuzzy keyword search and block-level deduplication. Finally, security analysis is given.

Key words: Data deduplication, outsourcing, privacy, keyword search

1 Introduction

Data deduplication is a technique for reducing the size of data in storage systems by detecting and eliminating redundant data and has been widely employed for data backup to minimize network and storage overhead. The system will perform duplicates for incoming data and does not record the data if it matches the existing stored data and refers other redundant data to that copy.

There have been a number of deduplication systems proposed with various deduplication strategies such as file-level or block-level deduplications depending on the size of deduplication. It can also be divided into inline deduplication system and post-process deduplication system, which performs deduplication before or after storing them respectively. Especially, with the advent of cloud storage, data deduplication techniques attracts more and more attention from both academic and industrial community. It has become one of critical techniques for the management of the ever-increasing volume of data in the information society. According to the analysis report of IDC, the volume of data in the world is expected to reach 40 trillion gigabytes in 2020 [2] with very high increasing speed. Today's commercial cloud storage services, such as Dropbox, Mozy, and Memopal, have been applying deduplication to save the network bandwidth and the storage cost with client-side deduplication.

To protect the confidentiality of outsourced data, the notion of convergent encryption [3] has been proposed. With the convergent encryption, the confidentiality of data can be achieved while realizing deduplication. In the deduplication system based on convergent encryption, the data will be encrypted/decrypted with a convergent key which is derived by computing the hash value of the content of data copy itself [3–5]. With this novel technique, identical data copies will generate the same convergent key and be encrypted into the same ciphertext, which allows the cloud to perform deduplication on the ciphertexts.

However, data encryption makes effective data utilization to be difficult given that there could be a large amount of outsourced data files. The keyword search over encrypted data in cloud computing has been proposed in recent years [6–12] and popularly used to selectively retrieve files of interest. Through this technique, both the data and keyword are encrypted such that privacy of

* Corresponding author.

A preliminary version of this paper has been presented at AINA2014 [1]
Email address: jinli71@gmail.com (Jin Li).

keyword information will be also achieved besides the confidentiality of data. In a searchable encryption scheme, an index will be built by the data owner for each file or each keyword, which is then uploaded to the cloud server with the encrypted data. By integrating the trapdoors of keywords within the index information, effective keyword search can be realized while both file content and keyword privacy are well-preserved.

Although such a technique allows the cloud server to perform search on behalf of user without knowing the information of the file or the keywords, the existing searchable encryption techniques have not been considered in a deduplication storage system with additional duplicate check and deduplication storage phases.

In this paper, we show how to construct secure deduplication systems with keyword search in cloud computing. We also formalize the security model for the encrypted keyword search in deduplication systems. Both the keyword privacy and content privacy can be achieved in the constructions. In our constructions, two kinds of index for keyword search have been proposed and analyzed. The integrity of the uploaded files could be verified by the users after downloading and decryption without any other traditional integrity auditing mechanisms. Such a property is achieved based on the technique of convergent encryption. The security analysis is also given under the proposed security model.

This paper is organized as follows. In Section 2, we present the system model and security requirements of the keyword search in deduplication storage system. In Section 3, the building blocks required in this paper are presented. Our constructions are described in Section 4 and Section 5. The security and performance analysis is given in Section 6. The related work is described in Section 7. Finally, we draw conclusions in Section 8.

2 Problem Formulation

2.1 System Model

In this paper, we consider a cloud storage system consisting of data owner, data user and cloud storage server. The files are assumed to be encrypted by the data owner before uploading to the cloud storage server. We assume the authorization between the data owner and users is appropriately done with some authentication and key-issuing protocols. After uploading the encrypted files, authorized users are allowed to perform the keyword search on these encrypted data with the aid of cloud server. In more details, an authorized user sends a request to selectively retrieve data files of his/her interest. Upon

receiving the request, the cloud storage server is responsible for pinpointing a set of files matching the searching request. All the matched file ID will be returned to the user as the searching result. Both client-side deduplication and server-side deduplication are supported in our system.

- *Data owner.* The data owner is an entity that outsources data storage to the storage server and access the data later. In a client-side data deduplication system, only the first data owner of a file needs to upload while the following data owners of the same file does not require to upload the duplicate data any more.
- *Users.* The entity of users in the deduplication systems makes registration at cloud storage server and has privileges to access some data files shared by some data owners.
- *Cloud storage server.* The cloud storage server is an entity that provides the data storage service for the users. Furthermore, the cloud storage server will also perform duplicate check before users upload their files. If there is identical content stored in cloud storage server, the users are not required to upload the file again, which can reduce the storage cost at the server side and save the upload bandwidth at user side.

Both file-level and block-level deduplications are supported in our system. To upload a file, a user first performs the file-level duplicate check. If the file is a duplicate, then all its blocks must be duplicates as well; otherwise, the user further performs the block-level duplicate check and identifies the unique blocks to be uploaded. Each data copy is associated with a tag for the duplicate check, which is computed with a hash function on the data file.

2.2 Adversary Model

We consider a semi-trusted cloud storage server in our deduplication storage system. To prevent the cloud storage server from getting sensitive information, data files are encrypted before uploading by the data owners. The cloud server will honestly follow our protocol and may try to derive other sensitive information from users' search requests. Thus, besides the confidentiality of data, the privacy of the keyword search should also be carefully be protected. In another word, the keyword search should be conducted in a secure manner that allows data files to be securely retrieved while revealing as little information as possible to the cloud server. For deduplication storage system supporting keyword search, we will follow the security definition deployed in the traditional searchable encryption [13] and deduplication system. More specifically, it is required that nothing should be leaked from the remotely stored files and index beyond the outcome and the pattern of search queries. Furthermore, we only consider the semantic security when the files

are unpredicable, that is, the files with high min-entropy. Actually, this requirement has recently been formalized in [14] and called the privacy against chosen distribution attack. This also implies that the data is secure against the adversary who does not own the data. The cloud storage server can obtain the knowledge of encrypted data and encrypted keyword in the search request. Its goal is to obtain some sensitive information from these obtained knowledge. To break the security of data from a data owner, the cloud storage server is able to collude with the other unauthorized users.

3 Preliminaries

In this section, some tools and notions are introduced in our construction, including the Bloom filter and trapdoors of keywords, message authentication code and convergent encryption. These preliminaries are utilized to design keyword search and encryption in this paper.

Bloom Filter A Bloom filter [15] is utilized to prove membership of a set $S = \{s_1, \dots, s_n\}$ with n elements. It is represented by an array of m bits, where each bit is initially set to 0. The filter uses r independent hash functions $h_1, \dots, h_r$, where $h_i : \{0, 1\}^* \rightarrow [1, m]$ for $i \in [1, r]$. For each element $s \in S$, the array bits at positions $h_1(s), \dots, h_r(s)$ are set to 1. To determine if an element w belongs to the set S , we check the bits at positions $h_1(w), \dots, h_r(w)$. w is considered to be a member of the set S if all the checked bits are 1's with some probability of a false positive. In particular, for keyword w with m -bit array Bloom filter, the probability of a false positive f is $f = (1 - (1 - 1/m)^{r|S_w|})^r \approx (1 - e^{-r|S_w|/m})^r$. False positives occur because each location may have also been set by some element other than w . On the other hand, if any checked bits are 0, then w definitely does not belong to S .

Trapdoors of Keywords Trapdoors of the keywords can be realized by applying a one-way function f , which could be defined as a hash function. Given a keyword w and a secret key sk , the trapdoor of w could be computed as $T_w = f(sk, w)$. The trapdoors of keywords constructed in this way achieves trapdoor privacy. It means that if a user who has not been assigned searching privilege, cannot get any information of the keywords underlying the search requests or index.

Message authentication code. A message authentication code (MAC) is a short piece of information used to authenticate a message and to provide integrity and authenticity assurances on the message. In our construction, the message authentication code is applied to achieve the integrity of the out-sourced stored files. It can be easily constructed with a keyed (cryptographic) hash function, which takes input a secret key and an arbitrary-length file to be

authenticated, and outputs a MAC. Only users with the same key generating the MAC can verify the correctness of the MAC value and detect if the file has been changed or not.

Convergent encryption Convergent encryption provides data confidentiality in deduplication. The encryption key, which is called convergent key here, is computed from data file. Then, the file is encrypted with such a convergent key with symmetric encryptions. To support deduplication, a tag is also computed for the file and used to detect duplicate by the cloud storage server. If two data copies are the same, then their tags are the same. Note that both the convergent key and the tag are independently derived, and the tag cannot be used to deduce the convergent key and compromise data confidentiality. Formally, a convergent encryption scheme can be defined with four primitive functions: 1) the key generation algorithm $\text{KeyGen}(M) \rightarrow K$ maps a data copy M to a convergent key K . We use a hash function $H_0(M) = K$ as the computation of convergent encryption key, 2) the symmetric encryption $\text{Enc}(K, M) \rightarrow C$ takes both the convergent key K and the data copy M as inputs and then outputs a ciphertext C , 3) the decryption algorithm $\text{Dec}(K, C) \rightarrow M$ takes both the ciphertext C and the key K as inputs and then outputs M , and 4) the tag generation algorithm can be defined as another hash function of M .

4 The Deduplication System Supporting Keyword Search

4.1 A Basic Construction

The main idea of the construction is based on traditional keyword search mechanism. Given a collection of N data files $\mathcal{F} = (F_1, F_2, \dots, F_N)$ stored in the cloud server by a user, a set of distinct keywords $W = \{w_1, w_2, \dots, w_p\}$ is defined for these files. Upon receiving a searching request w , the storage server will search and return a set of file IDs whose corresponding data files contain the word w , denoted as FID_w . That is, for each keyword w , the corresponding file ID set FID_w is stored. The main idea of keyword search for encrypted data is similar to the plaintext except that each keyword is replaced with $f(sk, w)$, where sk is the user's private key. Note that in this storage system, the user can only search the files belonging to himself with his own private key sk . In this section, we start with some straightforward solutions, the analysis of which will motivate us to develop more efficient and secure ones. Specifically, the scheme goes as follows:

System setup. Define the security parameter as 1^λ and initialize a convergent

encryption scheme. Assume there are N encrypted data files $\mathcal{C} = (C_{F_1}, C_{F_2}, \dots, C_{F_N})$ stored in the cloud server by a user, where $k_F = H_0(F)$ and $C_F = \text{Enc}_{\text{CE}}(k_F, F)$. The user also defines a set of distinct keywords $W = \{w_1, w_2, \dots, w_p\}$. The user also builds an index for these files as follows. For each keyword $w_i \in W$, the user computes trapdoor $T_{w_i} = f(sk, w_i)$ with secret key sk and attaches FID_{w_i} . Assume that $\{C_F, FID_{w_i}\}$ has been stored on the cloud by the user.

File Upload. To upload a file F , the user interacts the cloud storage server and tests if there is any duplicated copy stored in the cloud storage server. A file tag $\phi_F = H(F)$ will be computed and sent to the cloud server for the file duplicate check.

- If a duplicate is found by the storage server, the user will be assigned a pointer, which allows him to access the file.
- Otherwise, if no duplicate is found, the user computes the encrypted file $C_F = \text{Enc}_{\text{CE}}(k_F, F)$ with the convergent key $k_F = H_0(F)$ and uploads C_F to the cloud server. The user also encrypts the convergent key k_F with a symmetric encryption key sk and gets the ciphertext $C_{k_F} = \text{Enc}(sk, k_F)$, which is also uploaded to the cloud server.

After this phase, the user goes into the index building phase. That is, for each keyword w belonging to F , the data owner computes trapdoors $T_w = f(sk, w)$ with secret key sk . Then he uploads the trapdoors $\{T_w\}$ and identity of F to the cloud storage server. Upon receiving the updating request, the storage server updates the user's index by adding the trapdoor and corresponding file identity if there is no such a trapdoor; otherwise, the cloud server just adds the file identity to the FID_w .

Keyword search. Assume a user wants to search files containing keyword w . Then he first computes and sends a trapdoor for this keyword as $T(w) = f(sk, w)$ with his secret key sk . Upon receiving the search request, the cloud storage server first compares T_w with the index and returns the searching result as FID_w if there exists a match. Otherwise, a symbol $\perp$ is returned.

File Retrieving. Suppose a user wants to download a file F . It first sends a request and the file names to the cloud storage server. Upon receiving the request and file names, the cloud storage server will check whether the user is eligible to download the files. If failed, the cloud storage server sends back an **abort** signal to the user to indicate the download failure. Otherwise, the cloud server returns the corresponding ciphertexts $\{C_F\}$ and $\{C_{k_F}\}$. Upon receiving the encrypted data from the cloud server, the user decrypts and gets the keys $\{k_F\}$ by using sk stored locally. With the convergent keys, the user is able to recover the original files $\{F\}$. Then, the user also verifies the files by checking if $H_0(F) = k_F$. If yes, then the user accepts this file; otherwise, the file will be

rejected because it cannot pass the integrity verification.

4.2 Another Construction based on Bloom Filter

In this section, we show how to utilize the Bloom filter [15] to represent the keyword set for each file. More specifically, the trapdoor set $\{T_w\}_{w \in F}$ is inserted into file F 's Bloom filter as the index stored on the server. Now by binding the file identifiers FID to F 's Bloom filter, a per file index is generated to track the data files.

System setup. The description of system setup is similar to the basic construction. Assume there are N encrypted data files $\mathcal{C} = (C_{F_1}, C_{F_2}, \dots, C_{F_N})$ stored in the cloud server by a user with convergent encryption. The user also builds an index for these files as follows. For each file F , the user computes trapdoors $T_{w_i} = f(sk, w_i)$ with secret key sk for all keywords in this file. Then, these trapdoors are inserted into file F 's Bloom filter as the index stored on the server, binding with the file identity FID.

File Upload. To upload a file F , the user performs the duplicate check as the basic construction, which is omitted here. After the duplicate check phase, the user starts the index building phase. That is, the data owner computes trapdoors $T_w = f(sk, w)$ with secret key sk . Then he uploads these trapdoors and identity of F to the cloud storage server. After receiving these trapdoors, the cloud storage server inserts each of them into the Bloom filter for F and keeps the updated Bloom filter as the index for F .

Keyword search. Upon receiving the search request T_w from a user with secret key sk , the server tests which Bloom filters contain 1's in all r locations and returns the search results, where we assume that there are r independent hash functions $h_1, \dots, h_r$ used in the construction of Bloom filter.

Note that the search cost associated with this solution is $O(|N|)$, where N is the number of data files.

File Retrieving. The description of file retrieving is the same as the basic construction and we omit it here for simplicity.

5 Advanced Deduplication Storage Systems

5.1 A Two-Level Deduplication System

In a scalable system, the storage size for the index of keywords is also very huge, especially when the number of users and files is large. To utilize the deduplication with keyword search, it is critical to support the fuzzy search property. In the previous related work, there are many results that discussed the storage size of such keyword index. Furthermore, to realize the fuzzy keyword search with better utilization of outsourced data, this problem becomes even worse. The trivial straightforward approach by just listing all the possible keywords within some distance, apparently provides fuzzy keyword search over the encrypted files. However, this approach has serious efficiency disadvantages. The simple enumeration method in constructing fuzzy keyword sets would introduce large storage complexities, which greatly affect the usability. Recall that in the definition of edit distance, substitution, deletion and insertion are three kinds of operations in computation of edit distance. The numbers of all similar words of w_i satisfying $\text{ed}(w_i, w'_i) \leq d$ for $d = 1, 2$ and 3 are approximately $2k \times 26$, $2k^2 \times 26^2$, and $\frac{4}{3}k^3 \times 26^3$, respectively. When assume that the output length of hash function is 160 bits, the resulted storage cost for the index will be 30GB. Recently, Li et al. [6] propose another solution to construct efficient fuzzy keyword sets with smaller size. Even if this novel solution is efficient, for system with huge number of users, the storage overhead for these indexes is also very big. To support the scalability in a deduplication system supporting keyword search, we consider how to design a deduplication system with two-level deduplication. That is, the new system can not only support deduplication with file content, but also realize the deduplication for index.

Main idea. To protect the privacy of keyword from the keyword search request and index, the keywords are always pre-computed by combining some private key owned by the users, for example, the hash value on the private key and the keyword. Thus, different users with different private keys results different indexes. In the previous work, even the index can be the same for users in the same group with the same private key, they still have to upload these same values to the servers without deduplication technique. Thus, in this Section, we will show how to combine the keyword search with the deduplication in order to save the storage and uploading bandwidth. A private cloud server will be introduced, who will be in charge of the generation of all the trapdoor of keyword. In this way, the same trapdoor will be generated for all the users with the same keyword. Furthermore, the information for the keyword can be protected. The private cloud server is assumed to be trusted and honest.

Before giving the construction, some preliminaries are given below.

Edit Distance There are several traditional ways to quantitatively measure the string similarity. We will use the edit distance [16] as [6]. The edit distance $\text{ed}(w_1, w_2)$ between two words w_1 and w_2 is the number of operations required to transform one of them into the other. The three primitive operations are listed as follow, substitution: changing one character to another; deletion: deleting one character from a word; insertion: inserting a single character into a word. Based on the three primitives, we can define the following set for a word: Given a keyword w , let $S_{w,d}$ denote the set of words w' satisfying $\text{ed}(w, w') \leq d$ for a certain integer d .

Fuzzy Keyword Search Using edit distance, the definition of fuzzy keyword search can be formulated as follows: Given a collection of n encrypted data files $\mathcal{C} = (F_1, F_2, \dots, F_n)$ stored in the cloud server, a predefined set of distinct keyword set W , a provided search request (w, d) for w with a specified edit distance d , the execution of fuzzy keyword search returns a set of file IDs whose corresponding data files possibly contain w : if $w = w_i \in W$, then just return FID_{w_i} , which denotes the identifier containing w_i ; otherwise, if $w \notin W$, then return $\{FID_{w_i}\}$, where $\text{ed}(w, w_i) \leq d$. In other words, the system returns all the files containing the search request itself if there is an exact match among W ; otherwise, return all the possible files with the keywords falling within the specified edit distance. In fact, d can be different for distinct keywords and the system will return $\{FID_{w_i}\}$ satisfying $\text{ed}(w, w_i) \leq \min\{k, d\}$ if exact match fails.

Next, we describe the construction for the deduplication supporting fuzzy keyword search.

Similar to the above constructions, we also need to define the security parameter. We also assume that there are N encrypted data files $\mathcal{C} = (C_{F_1}, C_{F_2}, \dots, C_{F_N})$ stored in the cloud server by a user. For these files, there are a set of distinct keywords $W = \{w_1, w_2, \dots, w_p\}$. The private cloud server keeps two private keys sk, K , which will be used to compute the trapdoor of keyword and convergent keys.

Index Upload. In the index building phase, for each keyword $w_i \in W$, the user sends the keyword and corresponding file identifiers to the private cloud server. The private cloud server first computes the fuzzy keyword set $S_{w_i,d}$ and computes trapdoors $T_{w'_i} = f(sk, w'_i)$ for all $w'_i \in S_{w_i,d}$ with secret key sk . These trapdoors will be sent to cloud servers. The cloud servers first check if there is an identical fuzzy set stored. If there is, then, the corresponding index will not be stored again by the storage server; If there is only a subset that is identical with the values stored in the cloud, then, the duplicate will be only stored once. For the above cases, the corresponding file identifiers need

to be returned to the private cloud server, who will first decrypt to get the file identifiers and the new file identifiers will be added and updated. Finally, the updated encrypted file identifier is sent to the storage server. Otherwise, the private cloud server computes $\text{Enc}(sk, \text{FID}_{w_i} \| w_i)$ and outsources the index table $\{\{T_{w'_i}\}_{w'_i \in S_{w_i,d}}, \text{Enc}(sk, \text{FID}_{w_i} \| w_i)\}$.

File Upload. After uploading the index, the user will start to upload each file F . First, the user interacts with the cloud storage server. That is, a file tag $\phi_F = H(F)$ will be computed and sent to the private cloud server, who will sent back $\phi_{F,K} = H(H(F), K)$ to the user. This tag generation method is similar to [17], which is used to prevent from the brute-force attack launched by the public cloud server or dishonest users. Then, the tag value $\phi_{F,K}$ is sent to the storage server for duplicate check.

- If a duplicate is found by the storage server, the user will be assigned a pointer, which allows him to access the file.
- Otherwise, if no duplicate is found, the user computes the encrypted file $C_F = \text{Enc}_{\text{CE}}(k_F, F)$ with the convergent key $k_F = H(H_0(F), K)$ and uploads C_F to the cloud server. Note that the convergent key k_F is also generated by the private cloud server with his secret key K . The user also encrypts the convergent key k_F with a symmetric encryption key sk_u and gets the ciphertext $C_{k_F} = \text{Enc}(sk_u, k_F)$, which is also uploaded to the cloud server.

Keyword search. Assume that an authorized user types in w as the searching input, with the pre-set edit distance k . The input is first transformed to a fuzzy set $S_{w,k}$ based on the same rule utilized by the data owner above. Then, with this fuzzy set, the private cloud server will first compute the trapdoors $\{T_{w'}\}_{w' \in S_{w,k}}$ for each element $w' \in S_{w,k}$ with his private key sk . Then, the hash values are sent to the public cloud storage server as the searching request.

Upon receiving the searching request, the storage server first compares these hash values with the index and returns the search result as $\text{Enc}(sk, \text{FID}_w \| w)$ if there exists an exact match. The private cloud server first decrypts the ciphertext and then sends the file identifiers $\text{FID}_w \| w$ back to the user.

File Retrieving. After above searching request, suppose that a user wants to download a file F . It first sends a request and the file names to the cloud storage server. Upon receiving the request and file names, the cloud storage server will check whether the user is eligible to download the files. If failed, the cloud storage server sends back an **abort** signal to the user to indicate the download failure. Otherwise, the cloud server returns the corresponding ciphertexts $\{C_F\}$ and $\{C_{k_F}\}$. Upon receiving the encrypted data from the cloud server, the user decrypts and gets the keys $\{k_F\}$ by using sk_u stored locally. With the convergent keys, the user is able to recover the original files $\{F\}$. Then, the user also verifies the files by checking if $H_0(F) = k_F$. If yes, then

the user accepts this file; otherwise, the file will be rejected because it cannot pass the integrity verification.

5.2 Block-Level Deduplication Scheme

We only discuss the keyword search in file-level deduplication system in Section 4. However, it is easy to extend the above constructions into block-level deduplication supporting key word search. Note that the keyword search is still performed over files. In this section, we show how to achieve the keyword search in block-level deduplication. In a block-level deduplication system, the user should still firstly perform the file-level deduplication. The block-level deduplication will be performed if and only if no duplicate is found. In more details, the user divides this file into blocks and performs block-level deduplication. The system setup is the same as the file-level deduplication system, except the block size parameter will be defined additionally. Next, we give the details of the algorithms of **File Upload**.

File Upload. To upload a file F , the user first performs the file-level deduplication. If a duplicate is found, the user will perform the file-level deduplication as that in Section 4.

If no duplicate is found, the user performs the block-level deduplication as in [18] by dividing F into a set of fragments $\{B_i\}$. For each B_i , the user performs a block-level duplicate check by computing $H(B_i)$, where the data processing and duplicate check of block-level deduplication is the same as that of file-level deduplication if the file F is replaced with block B_i . Thus, we omit it here.

The traditional deduplication method cannot prevent illegal data distribution attack and side channel attack. To prevent the adversary from launching these attacks, a new concept of Proof of Ownership (PoW) has been proposed [19], which enables users to prove their ownership of data copies to the storage server. PoW is implemented as an interactive algorithm. The user first computes a short hash value $H(F)$ from a file F and sends it to the cloud storage server, who will challenge the user with some random chosen index values. If the user could correctly answer these challenges, the proof is passed. Thus, the PoW technique can be also applied in our construction to prevent the deduplication systems from these attacks. Through integrating PoW in our deduplication systems, we can achieve higher security.

6 Security and Performance Analysis

6.1 Security Analysis

In this section, we present the security analysis for the deduplication system with keyword search in Section 4. The security analysis for the other constructions is similar and thus omitted here. Some basic cryptographic tools have been applied to in our construction achieve the deduplication protocol. To show the security of this protocol, we assume that the underlying building blocks are secure, including the message authentication code scheme and the convergent encryption scheme. Thus, the security will be analyzed based on the above security assumptions.

Completeness. In our constructions, the cloud storage server is assumed to be honest and curious, which means that they will follow the protocols. Thus, if the data files have been successfully uploaded and stored at servers with correctly computed index, the data owner can retrieve them from the cloud storage server. The files can be decrypted by the data owner with convergent key based on the correctness of convergent encryption. Furthermore, the index consisting of trapdoors of keywords for each file. Thus, if a data owner submits a keyword search query to the cloud storage server, the exact match value will be returned to the user after performing the keyword search in the cloud server.

Search privacy. In our construction, the computation of index and request of the same keyword is identical as $f(sk, w)$ for a keyword w with secret key sk . The access pattern privacy will not be considered in our systems, and thus we will not take the frequency attack into our security analysis. The security of search privacy is proven with reduction. In more details, suppose the searchable encryption scheme fails to achieve the search privacy against the chosen keyword. It means there exists adversary who can get the underlying information of keyword from the index. However, as we know, the trapdoor is computed from a message authentication code. Then, based on such an adversary, we are able to build an algorithm that utilizes the adversary to break the security of message authentication code. This algorithm has an access to an oracle that can answer the message authentication code on any message m . The adversary can query the trapdoors for a set of keywords, which can be simulated by our designed algorithm by querying the message authentication code oracle. After making these trapdoor queries, the adversary outputs two challenge keywords w_0 and w_1 . Our designed algorithm just chooses one of them and sends it to the challenger to compute a message authentication code. This challenge value will be sent back to the adversary. If the adversary can effectively guess which keyword is computed in the trapdoor, then

our designed algorithm can also use this guess as the output to the message authentication code and break the security of the underlying message authentication code. Therefore, based on the assumption of the security of message authentication code, the adversary cannot get any valid information from the trapdoor under the chosen trapdoor attack. Based on the above analysis, we can get the security result of search privacy for our constructions.

For the construction of deduplication supporting fuzzy keyword search, the technique in [6] is used, which has been proven secure with respect to the keyword privacy under the corresponding security model. Thus, the searching privacy can be achieved in a similar way. Note that in our security model, the private cloud server is assumed to be trusted. It owns the private key and all the keywords have been sent to the private cloud server for the generation of trapdoors. Of course, even if the private cloud server is not honest, only the privacy of keyword will be leaked and the file confidentiality is still kept. Thus, in this weaker security model, the security can be achieved similar to the traditional deduplication system.

File confidentiality. In the deduplication system, all the files have been encrypted by using the convergent encryption scheme. Furthermore, two kinds of different encryption methods have been applied in our two constructions. In the first construction, the data is encrypted with the traditional convergent encryption scheme. The data encrypted with such encryption method cannot achieve semantic security as it is inherently subject to brute-force attacks that can recover files falling into a known set. The detailed security analysis on the convergent encryption can be found in many related works, which is omitted here.

We present the security analysis for the construction with private cloud server. In this construction, the file content is not only computed by using the convergent encryption key, but also the private key owned by the private cloud server is embedded. In this way, the ciphertext is unpredictable even if the original file is predictable. Furthermore, for the security analysis, two kinds of adversaries, including external adversaries and internal adversaries, are analyzed. For the internal adversaries, they are provided with some convergent encryption keys additionally. However, these convergent encryption keys have no security impact on the data confidentiality because these convergent encryption keys are computed with secret key K . And the secret key K is protected by the private cloud server. Furthermore, the convergent key is encrypted by using the symmetric encryption with a key chosen randomly by the user. Thus, based on the above analysis, the convergent key in our construction is not deterministic in terms of the file, which still depends on the secret key stored by the private cloud server and unknown to the adversary. As a result, if the private cloud server is honest and does not collude with the adversary, the confidentiality of file is semantically secure for both predictable and unpredictable file.

6.2 Performance Analysis

In this section, we show the evaluation for the efficiency of the given constructions. The deduplication protocol is very similar to the previous related work [18]. Thus, we only focus on the keyword search efficiency in this protocol. Two kinds of time costs have been analyzed, that is, the time cost for the construction of fuzzy keyword set and the time cost for fuzzy keyword search. Our experiments are simulated with python 3.4. Note that in order to precisely measure the overhead at both server and user sides, All experiments were performed on the same machine with Intel Core i5-3470 3.20 GHz CPU and 4G memory running Linux.

Fig. 1. Execution time of Fuzzy Search

Two edit distances have been chosen for the experiment, that is, $d = 1$ and $d = 2$. From the experiment, we can find that the search efficiency is very efficient for both edit distance. From these two figure, we can find that the cost of constructing fuzzy keyword set under $d = 1$ is much less than the case of $d = 2$. The reason is that it has much less elements in the first case. Actually, the efficiency can also be improved through the advanced techniques such as trie-traverse search scheme, where a multi-way tree can be built for storing the fuzzy keyword set. In more details, all trapdoors sharing a common prefix may have common nodes. Based on this observation, the root of such as tree is associated with an empty set and the trapdoor can be recovered in a search from the root to the leaf that ends the trapdoor. Obviously, any request for keyword search can be found by a depth-first search in this tree. The file identifiers, which have been encrypted, will be indexed according to their names. Finally, these index information will be stored at the ending node of the corresponding path. We also evaluate the search performance as the number of keywords increases in Fig. 1, which demonstrates that the average search time is very efficient. And based on the experimental results, the edit distance and the number of keywords affect the searching efficiency. Based on the theoretical results, the time for the construction of fuzzy set and

searching grows with the number of the keywords and edit distance. However, because of the utilization of fuzzy keyword set construction [6], the efficiency is practical for applications. We can also get the result on the efficiency for the exact match, which is more efficient than that of fuzzy search. All the results validate our efficiency analysis and demonstrate that our proposed constructions are very efficient and effective supporting fuzzy keyword search over encrypted data in cloud computing.

7 Related Work

Data deduplication techniques are very interesting techniques that are widely employed for data backup in enterprise environments to minimize network and storage overhead by detecting and eliminating redundancy among data blocks. There are many deduplication schemes proposed by the research community. Furthermore, deduplication is also widely adopted in practice such as Bitcasa, Ciphertite and Dropbox. Convergent encryption [3] ensures data privacy in deduplication, which was proposed for this special application without rigorous security analysis. There are also several implementations of convergent implementations of different convergent encryption variants for secure deduplication (e.g., [20–23]). Recently, Bellare et al. [14] formalized this primitive as message-locked encryption, presented detailed and formal security analysis for this notion. They defined several new security definitions and adversary model for this primitive and explore its application in space-efficient secure outsourced storage. The constructions described in [14] has been applied by some commercial cloud storage providers, such as Bitcasa, also deploy convergent encryption [14]. To further address the key-management problem in deduplication system, Li et al. [18] proposed a new deduplication system by distributing these keys across multiple servers after encrypting the files. The storage of these convergent keys also uses the technique of deduplication such that the storage and key management overhead at user side is reduced. Because of the weakness in convergent encryption, that is, the key is derived directly and determinedly from the message itself, Bellare et al. [24] proposed another method to improve the security by transforming the predictable message into unpredictable message. In their system, a random secret key is further added into the generation of file tag. In this way, the predicate files will be unpredictable because of this random number. Stanek et al. [25] presented a novel encryption scheme that provides differential security for popular data and unpopular data. For popular data that are not particularly sensitive, the traditional conventional encryption is performed. Another two-layered encryption scheme with stronger security while supporting deduplication is proposed for unpopular data. In this way, they achieved better tradeoff between the efficiency and security of the outsourced data.

There are also some work that present security attack on the deduplication system based on the traditional convergent encryption. Harnik et al. [26] presented a number of attacks that can lead to data leakage in cloud storage system supporting client-side deduplication. They observed that only a hash value has been used in these system to realize the ownership proof to the cloud storage server. As a result, the adversary can share these short values with other users and realize the content distribution attack. To prevent these attacks, Halevi et al. [19] proposed the notion of “proofs of ownership” (PoW) for deduplication systems, such that a client can efficiently prove to the cloud storage server that he owns a file without uploading the file itself. The technique is to use more entropy and longer proof to prevent the sharing. Several PoW constructions based on the Merkle Hash Tree are proposed [19] to enable client-side deduplication, which include the bounded leakage setting. Pietro and Sorniotti [27] propose another efficient PoW scheme by choosing the projection of a file onto some randomly selected bit-positions as the file proof. Xu et al. [28] also consider the data privacy and presented another scheme in a bounded leakage setting. In order to prevent from brute-force attack, Li et al. [17] proposed the notion of deduplication with privileges in a hybrid cloud architecture. In this system, only authorized users are able to send the valid request on the files with corresponding access policy.

Searchable encryption [11,29] has been widely studied in the context of cryptography. Among those works, most are focused on efficiency improvements and security definition formalizations. The first construction of searchable encryption was proposed by Song et al. [9], in which each word in the document is encrypted independently under a special two-layered encryption construction. Goh [11] proposed to use Bloom filters to construct the indexes for the data files. For each file, a Bloom filter containing trapdoors of all unique words is built up and stored on the server. To search for a word, the user generates the search request by computing the trapdoor of the word and sends it to the server. Upon receiving the request, the server tests if any Bloom filter contains the trapdoor of the query word and returns the corresponding file identifiers. To achieve more efficient search, Chang et al. [30] and Curtmola et al. [13] both proposed similar “index” approaches, where a single encrypted hash table index is built for the entire file collection. In the index table, each entry consists of the trapdoor of a keyword and an encrypted set of file identifiers whose corresponding data files contain the keyword. As a complementary approach, Boneh et al. [31] presented a public-key based searchable encryption scheme, with an analogous scenario to that of [9]. In their construction, anyone with the public key can write to the data stored on the server but only authorized users with the private key can search. As an attempt to enrich query predicates, conjunctive keyword search, subset query and range query over encrypted data, have also been proposed in [29]. Note that all these existing schemes support only exact keyword search, and thus are not suitable for Cloud Computing. Li et al. [6] presented a search scheme supporting fuzzy

keyword match, which is critical in practical applications. They proposed two methods to realize the fuzzy set for a keyword. However, all these works have only considered the traditional keyword search over encrypted files without supporting deduplication.

8 Conclusion

In this paper, for the first time we formalize and solve the problem of supporting efficient yet privacy-preserving keyword search for encrypted data in deduplication storage system. The convergent encryption has been utilized to protect the confidentiality of sensitive data while supporting deduplication. This paper firstly proposed two keyword search constructions in this scenario with different index formats. The integrity of the uploaded files could also be verified by the users after downloading and decryption without any other traditional integrity auditing mechanisms. Furthermore, two advanced constructions on the deduplication supporting fuzzy keyword search and security analysis demonstrates that our keyword search schemes are secure in terms of the definitions specified in the proposed security model.

Acknowledgement

This work was supported by National Natural Science Foundation of China (No.61100224, No. 61472091 and No.61272455), Natural Science Foundation of Guangdong Province (Grant No. S2013010013671), the Guangzhou Zhujiang Science and Technology Future Fellow Fund (Grant No. 2012J2200094).

References

- [1] J. Li, X. Chen, F. Xhafa, and L. Barolli, "Secure deduplication storage systems with keyword search," in *AINA*, 2014, pp. 971–977.
- [2] J. Gantz and D. Reinsel, "The digital universe in 2020: Big data, bigger digital shadows, and biggest growth in the far east," <http://www.emc.com/collateral/analyst-reports/idc-the-digital-universe-in-2020.pdf>, Dec 2012.
- [3] J. R. Douceur, A. Adya, W. J. Bolosky, D. Simon, and M. Theimer, "Reclaiming space from duplicate files in a serverless distributed file system." in *ICDCS*, 2002, pp. 617–624.

- [4] A. P. Sabzevar and J. P. Sousa, "Authentication, authorisation and auditing for ubiquitous computing: a survey and vision," in *IJSSC*, ser. DOI: 10.1504/IJSSC.2011.039107, 2014.
- [5] M. Yuriyama and T. Kushida, "Integrated cloud computing environment with it resources and sensor devices," in *IJSSC*, ser. DOI: 10.1504/IJSSC.2011.040342, 2014.
- [6] J. Li, Q. Wang, C. Wang, N. Cao, K. Ren, and W. Lou, "Fuzzy keyword search over encrypted data in cloud computing," in *INFOCOM*, 2010, pp. 441–445.
- [7] J. Li and X. Chen, "Efficient multi-user keyword search over encrypted data in cloud computing," in *Computing and Informatics*, vol. 32(4), 2013, pp. 723–738.
- [8] M. Bellare, A. Boldyreva, and A. O'Neill, "Deterministic and efficiently searchable encryption," in *Proceedings of Crypto 2007, volume 4622 of LNCS*. Springer-Verlag, 2007.
- [9] D. Song, D. Wagner, and A. Perrig, "Practical techniques for searches on encrypted data," in *Proc. of IEEE Symposium on Security and Privacy'00*, 2000.
- [10] A. Nagata, S. Yamamura, and M. Tsuru, "Data transfer exploiting multiple heterogeneous challenged networks-implementation and application," in *IJSSC*, ser. DOI: 10.1504/IJSSC.2012.047468, 2014.
- [11] E.-J. Goh, "Secure indexes," Cryptology ePrint Archive, Report 2003/216, 2003, <http://eprint.iacr.org/>.
- [12] F. Bao, R. Deng, X. Ding, and Y. Yang, "Private query on encrypted data in multi-user settings," in *Proc. of ISPEC'08*, 2008.
- [13] R. Curtmola, J. A. Garay, S. Kamara, and R. Ostrovsky, "Searchable symmetric encryption: improved definitions and efficient constructions," in *Proc. of ACM CCS'06*, 2006.
- [14] M. Bellare, S. Keelveedhi, and T. Ristenpart, "Message-locked encryption and secure deduplication," in *EUROCRYPT*, 2013, pp. 296–312.
- [15] B. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Communications of the ACM*, vol. 13, no. 7, pp. 422–426, 1970.
- [16] V. Levenshtein, "Binary codes capable of correcting spurious insertions and deletions of ones," *Problems of Information Transmission*, vol. 1, no. 1, pp. 8–17, 1965.
- [17] J. Li, Y. Li, X. Chen, P. Lee, and W. Lou, "A hybrid cloud approach for secure authorized deduplication," in *IEEE Transactions on Parallel and Distributed Systems*, 2014, p. Doi.ieeecomputersociety.org/10.1109/TPDS.2014.2318320.
- [18] J. Li, X. Chen, M. Li, J. Li, P. Lee, and W. Lou, "Secure deduplication with efficient and reliable convergent key management," in *IEEE Transactions on Parallel and Distributed Systems*, vol. 25(6), 2013, pp. 1615–1625.

- [19] S. Halevi, D. Harnik, B. Pinkas, and A. Shulman-Peleg, “Proofs of ownership in remote storage systems.” in *ACM Conference on Computer and Communications Security*, Y. Chen, G. Danezis, and V. Shmatikov, Eds. ACM, 2011, pp. 491–500.
- [20] P. Anderson and L. Zhang, “Fast and secure laptop backups with encrypted de-duplication,” in *Proc. of USENIX LISA*, 2010.
- [21] Z. Wilcox-O’Hearn and B. Warner, “Tahoe: the least-authority filesystem,” in *Proc. of ACM StorageSS*, 2008.
- [22] A. Rahumed, H. C. H. Chen, Y. Tang, P. P. C. Lee, and J. C. S. Lui, “A secure cloud backup system with assured deletion and version control,” in *3rd International Workshop on Security in Cloud Computing*, 2011.
- [23] M. W. Storer, K. Greenan, D. D. E. Long, and E. L. Miller, “Secure data deduplication,” in *Proc. of StorageSS*, 2008.
- [24] M. Bellare, S. Keelveedhi, and T. Ristenpart, “Dupless: Server-aided encryption for deduplicated storage,” in *USENIX Security Symposium*, 2013.
- [25] J. Stanek, A. Sorniotti, E. Androulaki, and L. Kencl, “A secure data deduplication scheme for cloud storage,” in *Technical Report*, 2013.
- [26] 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.
- [27] R. D. Pietro and A. Sorniotti, “Boosting efficiency and security in proof of ownership for deduplication.” in *ACM Symposium on Information, Computer and Communications Security*, H. Y. Youm and Y. Won, Eds. ACM, 2012, pp. 81–82.
- [28] J. Xu, E.-C. Chang, and J. Zhou, “Weak leakage-resilient client-side deduplication of encrypted data in cloud storage,” in *ASIACCS*, 2013, pp. 195–206.
- [29] D. Boneh and B. Waters, “Conjunctive, subset, and range queries on encrypted data,” in *Proc. of TCC’07*, 2007, pp. 535–554.
- [30] Y.-C. Chang and M. Mitzenmacher, “Privacy preserving keyword searches on remote encrypted data,” in *Proc. of ACNS’05*, 2005.
- [31] D. Boneh, G. D. Crescenzo, R. Ostrovsky, and G. Persiano, “Public key encryption with keyword search,” in *Proc. of EUROCRYPT’04*, 2004.