

Article

# MinHash-Based Fuzzy Keyword Search of Encrypted Data across Multiple Cloud Servers

Jingsha He, Jianan Wu , Nafei Zhu \* and Muhammad Salman Pathan 

Faculty of Information Technology, Beijing University of Technology, Beijing 100124, China; jhe@bjut.edu.cn (J.H.); wjnfighting@163.com (J.W.); salman@emails.bjut.edu.cn (M.S.P.)

\* Correspondence: znf@bjut.edu.cn

Received: 14 March 2018; Accepted: 24 April 2018; Published: 1 May 2018



**Abstract:** To enhance the efficiency of data searching, most data owners store their data files in different cloud servers in the form of cipher-text. Thus, efficient search using fuzzy keywords becomes a critical issue in such a cloud computing environment. This paper proposes a method that aims at improving the efficiency of cipher-text retrieval and lowering storage overhead for fuzzy keyword search. In contrast to traditional approaches, the proposed method can reduce the complexity of Min-Hash-based fuzzy keyword search by using Min-Hash fingerprints to avoid the need to construct the fuzzy keyword set. The method will utilize Jaccard similarity to rank the results of retrieval, thus reducing the amount of calculation for similarity and saving a lot of time and space overhead. The method will also take consideration of multiple user queries through re-encryption technology and update user permissions dynamically. Security analysis demonstrates that the method can provide better privacy preservation and experimental results show that efficiency of cipher-text using the proposed method can improve the retrieval time and lower storage overhead as well.

**Keywords:** cloud storage; multi-server; multi-user; fuzzy keyword search

## 1. Introduction

With the advancement of cloud computing, more and more enterprises and individuals choose the option of storing their data in cloud servers to enjoy the on-demand high quality applications and services from a shared pool of configurable computing resources as well as to reduce the burden of managing their own storage. Cloud, as an honest-but-curious platform, on the other hand, may pose a significant threat to privacy of user data. To prevent sensitive data from being leaked, user data should be encrypted before being outsourced to the cloud servers. However, data encryption would make the utilization of user data a great challenge due to the need of retrieving a large amount of cipher-texts as compared to plain-texts that can be easily identified through keyword search. In addition, users may sometimes be interested only in retrieving some specific files from the cloud, making keyword-based search a primary tool to access cloud data. Thus, it has become a serious issue to find the required files from a large number of cipher-text documents.

For the retrieval of cipher-text files, a lot of research has been done on fuzzy keyword search over encrypted data in the cloud. Song et al. proposed an encryption-searchable method in which an encryption keyword search scheme was implemented based on symmetric encryption system for cipher-text retrieval [1]. This scheme uses pseudorandom function, deterministic encryption and XOR algorithm to construct an encrypted keyword index and relies on the function of keyword search for encrypted data located on the server side to ensure that cipher-text information is not exposed. Chang et al. improved the above scheme through building a corresponding index for each document to improve the efficiency of retrieval, where each index contains the keyword threshold information of

a document [2]. Wang et al. proposed a keyword order search scheme, mainly utilizing the technique of order preserving encryption to protect the relevance score as well as achieving the exact sort of search results [3]. Cao et al. introduced the vector space model and the secure KNN (secure K-Nearest Neighbor) method in which a document vector is constructed for each document, a reversible matrix is used with document vector multiplication to encrypt the document vector and the similarity between the index vector and the search vector inner product is calculated to achieve multiple keyword ranked search over encrypted data in the cloud [4].

The above solutions only support the exact keyword search over encrypted data. However, it is common to see typos and misspellings in practical scenarios, making it necessary to develop solutions that can handle this issue in real applications. Li et al. proposed a wildcard-based fuzzy set construction method that takes wildcards into consideration in the formation of a fuzzy set whose size may be drastically large [5]. Suresh also proposed a fuzzy set construction method based on Grams with more space efficiency [6] in which a symbolic index tree is used to improve the efficiency of search. Liu et al. improved the above method by constructing a dictionary-based fuzzy set to reduce the size of index [7], but the method suffers from the problem of search accuracy. Wang et al. later proposed to use wildcard and index tree to make fuzzy search more efficient [8]. Afterwards, the verifiable fuzzy search scheme was improved by extracting the path information of the index tree structure. However, to find the most similar keywords, both schemes need to construct a fuzzy keyword set, which not only incurs heavy computation load and communication overhead, but also costs a large amount of storage space in the cloud servers. Besides, these schemes can only sort the search results by editing distance with rough results without being able to return accurate search results.

Compared to the single user model, the multi-user models for searchable encryption have become more practical in actual cloud storage environments because they allow more than one authorized users to share data files that are outsourced to the cloud. Zirtol et al. proposed a multiple user searchable encryption scheme based on the general access structure [9] that allows any user to add encrypted data to the outsourced database and any authorized user to retrieve and decrypt cipher-texts. Li and Chen proposed a searchable encryption scheme based on mixed structure, which could provide accurate keyword retrieval and fine-grained access control on encrypt data [10]. However, the scheme makes it possible for the server to obtain keyword information, which is not regarded as a secure practice.

In this paper, we propose a fuzzy keyword search scheme for the scenarios of multiple servers and multiple users that has the property of preserving security and privacy of data. Our contribution in this work can be summarized as follows:

- The proposed multiple server searchable encryption scheme can provide the functionality of cipher-text retrieval for multiple servers. Compared to the single server model, our method can offer the capability of processing big data while securing data in the cloud.
- The proposed scheme provides more flexible control of user access rights on cipher-text files. Access rights of different users on the same file are not required to be the same. Moreover, the scheme supports dynamic updating of user access rights to provide better security over shared data in the cloud.
- The proposed scheme can reduce the complexity of Min-Hash by eliminating the construction of the fuzzy keyword set by performing fuzzy keyword search using Min-Hash fingerprints. The scheme utilizes Jaccard similarity to order the results of retrieval, which not only reduces the amount of calculation for similarity, but also lowers the space overhead to achieve high efficiency of retrieval.
- Through constructing an efficient fuzzy keyword index tree, the proposed scheme uses the Min-Hash to generate the fingerprint index for the keywords without having to set the index storage space and define dictionary library in advance, which greatly reduces the complexity of search and saves a lot of storage space.

The rest of this paper is organized as follows. Section 2 introduces some related work on searchable encryption. Section 3 describes the proposed scheme, which includes the system model, the design goal, and a detailed description of the scheme. Section 4 contains the security analysis and provides the experiment results of the proposed scheme. Finally, Section 5 concludes the paper.

## 2. Related Work

Song et al. were among the first who put forward the notion of searchable encryption and proposed the encrypted keyword search scheme (SWP scheme) based on symmetric cryptography, which caught the attention of the academic community on the research of searchable encryption technology. In the SWP scheme, each word in the files is encrypted in a double encryption fashion using a specially designed structure. Whenever a user wants to search a file, the cipher-text of the keywords should be generated and sent to the server. Through scanning and contrasting the cipher-text using the encrypted keywords, the server can confirm the authentic keywords and count the number of occurrences. Thus, the overhead of search is proportional to the size of the database, causing the efficiency to be low.

Goh et al. proposed a scheme to improve the efficiency of encrypted search that uses the Bloom classifier to perform screen and preprocess for the encrypted search [11]. The scheme would construct a Bloom classifier through keyword index and use the classifier as the filter during preprocessing. Through a series of Hash operations, some invalid query thresholds can be excluded, thus avoiding invalid thresholds from complicated encrypted queries. The computational complexity of a search in the scheme can be reduced to be in direct proportion to the number of encrypted files. In addition, the security definition of IND-CKA (adaptive Chosen Keyword Attack) was also provided in the work. Chang et al. proposed a scheme that is very similar to the above scheme without using the Bloom filter [12].

Curtmola et al. improved the definition of security by considering the trapdoor issues and used the inverted index for the first time to improve the efficiency of retrieval [13]. In this scheme, a hash index table is set up for the entire encrypted files each record of which contains the trapdoor keyword information and the corresponding encrypted file address set that contains the keyword. The cost of retrieval was shown to be reduced to be proportional to the number of keywords, not the number of files in the database. Besides, the SSE-2 scheme proposed in the work could support multi-user search, but only the data owners are allowed to upload encrypted data. Curtmola et al. also proposed a method for symmetric searchable encryption. Through using a pseudo-random function and an array structure, the scheme would construct a new file index structure to optimize the query mechanism and provide fast retrieval of encrypted data.

The development of public key cryptography has helped keyword search to meet new requirements for applications. Boneh developed a public key-based keyword search scheme (PEKS), defined the corresponding security model, successfully proved the security of the scheme, and applied the encryption search technology in public key cryptosystems [14]. Li et al. proposed a scheme based on BF-IBE, a public key encryption with keyword index and the trapdoor retrieval through the linear Hash function, and showed that the method could resist keyword attacks during query processing [15], ensuring that it is not possible for the server to obtain any information about the encrypted keyword through using Hash as well as the bilinear function.

It should be noted that the public key encryption with keyword search scheme introduced above is designed for single user environments. The user, as the owner of the data, owns the private key of the query and only the user can query the encrypted database stored in the server. In the cloud data sharing model, since data owners can authorize other users to share data resources, it would be too hard for single users to manage user access privileges in such encrypted keyword search schemes. There is thus the need to develop multi-user encryption search schemes that would allow dynamic authorization or revocation of user access privileges. In recent years, some research has been done on the development of multiple user encryption key search schemes in the cloud environment [16] that

can support multi-user encryption keyword search through using linear pair and proxy re-encryption technologies. Yang et al. introduced the idea of proxy re-encryption into public key searchable encryption and used query trapdoor conversion on the server side to allow multiple users to perform keyword query through query trapdoor generated by different keys [17]. A user authorization and revocation system was also designed. The multiple keyword search was mostly focused on the logic with connectives support. Bijral et al. proposed a ranking search algorithm that can also protect privacy through encrypting the frequency of keywords using an order preserving encryption algorithm [18].

To improve the utilization of data, Li et al. proposed a fuzzy keyword search scheme that can still return relevant data in case that the user input had minor errors or format inconsistencies [19]. The scheme makes use of two fuzzy set construction methods based on grams and wildcards, respectively, and defines the similarity between keywords based on the edit distance. Wang et al. performed further research on fuzzy keyword search with formal proof of security [20].

Due to security and performance considerations, it has become increasingly likely that the owner of big data wants to spread massive amount of data across multiple cloud servers, making it necessary to develop solutions for the multi-server model. This paper will propose a fuzzy keyword search scheme for multiple servers and multiple users that can preserve the security and privacy and, at the same time, improve the efficiency of data retrieval.

### 3. The Proposed Fuzzy Keyword Search Scheme Based on MinHash

#### 3.1. Preliminaries

LSH (Locality Sensitive Hashing) was first proposed by Indyk et al [21] to solve the problem of approximate nearest neighbor search. The basic idea is to generate the same hash key value for the points that are close in terms of their distance in the dataset. It has been shown that LSH algorithm has the obvious advantages of lowering the consumption of space and improving the efficiency of query processing compared to other schemes. The LSH method can thus help to return approximate nearest neighbor query results quickly in a probabilistic way, ensuring the accuracy of query results as well as the efficiency in terms of space and time.

LSH provides an effective way of retrieving one or more data points that are adjacent to the query data point in a massive and high dimensional dataset. LSH can not only find the most adjacent data to the query point while minimizing the number of data points that need to be matched, but also ensure a high probability of finding the nearest neighbor data points. As one of the most commonly used methods of LSH, the MinHash algorithm uses Jaccard similarity as a measurement standard to detect the similarity of two objects. The higher the Jaccard similarity, the more similar the two objects. In addition, the smaller the distance between two objects, the greater the possibility that two objects belong to the same class. Traditional methods are not very efficient in measuring the similarity between two datasets with a large number of elements, i.e., the dimension of the feature space is very large. In such cases, MinHash can deal with the problem with great efficiency.

##### 3.1.1. Jaccard Similarity

Jaccard similarity, also known as Jaccard coefficient, is a measure of similarity based on the metrics of distance [22]. The definition of Jaccard coefficient is as follows:

**Definition 1 (Jaccard coefficient).** Given two finite sets  $A$  and  $B$ , the Jaccard coefficients of  $A$  and  $B$   $J(A, B)$  is defined by the following equation:

$$J(A, B) = |A \cap B| / |A \cup B| \quad (1)$$

As can be seen in Equation (1), the Jaccard similarity of sets  $A$  and  $B$  is equal to the number of common elements of the two sets divided by the total number of elements in them. It is clear that Jaccard similarity falls into the range  $[0, 1]$ .

**Definition 2 (Jaccard coefficient).** Jaccard similarity (A, B) can convert to Jaccard distance using the following equation:

$$\text{Jaccard Distance (A, B)} = 1 - J(A, B) \tag{2}$$

It is thus clear from Equation (2) that the bigger the Jaccard similarity, the more similar the two objects that are involved and the smaller the distance between the two objects.

To derive the similarity of object sets, we can use a feature matrix in which each column represents a set and each row expresses the same element of all the object sets. If an element appears in the corresponding row of a set, then the value of the row is set to 1, otherwise, it is set to 0. For example, for the set S = {a, b, c, d, e} in which the four object sets are S<sub>1</sub> = {a, d}, S<sub>2</sub> = {c}, S<sub>3</sub> = {b, d, e} and S<sub>4</sub> = {a, c, d}, the matrix of the four object sets is shown in Table 1.

**Table 1.** The feature matrix of four object sets.

Element	S <sub>1</sub>	S <sub>2</sub>	S <sub>3</sub>	S <sub>4</sub>
a	1	0	0	1
b	0	0	1	0
c	0	1	0	1
d	1	0	1	1
e	0	0	1	0

Using Equation (1), the Jaccard similarity of sets S<sub>1</sub> and S<sub>4</sub> is 2/3 in the above example.

**Definition 3 (MinHash Signature).** Given a random sequence A of n elements, the MinHash signature  $h_{\min}(A)$  of the sequence is defined as follows:

$$h_{\min}(A) = \min_{j \in A} \pi(j) \tag{3}$$

**Definition 4 (MinHash Function).** Given two finite sets A and B, the probability that the two sets have the same value of MinHash is defined as:

$$\Pr\{h_{\min}(A) = h_{\min}(B)\} = J(A, B) \tag{4}$$

In applications, the MinHash function generates multiple hash for the target items so that similar items are more likely to hash into the same bucket than dissimilar ones. Moreover, if a document with at least one element being hashed into the bucket is considered to be a candidate, similarity between candidates can be examined to find similar documents in the candidate set.

For two sets A and B,  $h_{\min}(A) = h_{\min}(B)$  means that any element with its MinHash value in  $A \cup B$  is also in  $A \cap B$ . The hypothesis is that  $h(x)$  is a good Hash function if it has good homogeneity since it can map different elements to different integers. Therefore, the similarity of sets A and B is the same as the probability that the MinHash values of A and B are the same after passing through the Hash. Furthermore, the probability that they become the candidate pair is  $1 - (1 - S^L)^K$ , where S is the Jaccard similarity of sets A and B.

### 3.1.2. Order Preserving Encryption

Order preserving encryption (OPE) [23] is a kind of encryption that would keep the encrypted data elements in the same sequential order as that of the original data elements before encryption. With OPE, comparison of ciphertext data can be realized. For example, if  $a < b$ , then  $OPE(k, a) < OPE(k, b)$ , where k is the key used for the encryption.

### 3.1.3. Definition of Parameters

Following are a list of parameters that will be used throughout our discussion.

- $F = (f_1, f_2, \dots, f_k)$ : the set of plaintext documents;
- $U = (u_1, u_2, \dots, u_m)$ : the group of authorized users;
- $C = (c_1, c_2, \dots, c_k)$ : the set of ciphertext documents corresponding to  $F$ ;
- $W = (w_1, w_2, \dots, w_n)$ : the set of keywords extracted from  $F$ ;
- $FID_t = (FID_1, FID_2, \dots, FID_t)$ : the set of identifiers for a document;
- $S_i$ : the fingerprint of keywords  $w_i$ ;
- Score: the score of the correlation degree of keywords;
- $ID_{w_i}$ : the set of addresses for a document that contains the keyword  $w_i$ ;
- $T_w$ : the trapdoor of the keyword  $w$ ;
- $\Delta = \{\alpha_i\}$ : a predefined set of symbols, where  $|\Delta| = 2^n$  and each symbol can be expressed by  $n$  bits;
- $G_W$ : the index tree that contains all the fuzzy keywords;
- $h(hk, \bullet)$ : one-way hash function using key  $hk$ ;
- $(sk, \bullet)$ : the symmetric encryption algorithm using key  $sk$ ;
- $OPE(ek, \bullet)$ : an order-preserving encryption function using key  $ek$ ;
- $D(w) = \{id(D_i) | w \in D_i, 1 \leq i \leq n\}$ : the collection of IDs of all the documents containing the keyword  $w$ .

### 3.2. The System Model

In our proposed model, searchable encryption involves four different parties, as illustrated in Figure 1: the data owner, the authorized user, the server  $S$  and the cloud servers  $S_1, S_2, \dots, S_N$ . The data owner encrypts data before uploading it into the cloud as well as authorizes users to retrieve it. After authorization, the user can retrieve the corresponding cipher-text files according to keywords. The server  $S$  is dedicated to store the index after the data owner encrypts it while providing partial retrieval services to the user. The cloud servers  $S_1, S_2, \dots, S_N$  store the cipher-text files uploaded by the data owner and the cipher-text index by the server  $S$  after re-encryption while providing retrieval services to the user.

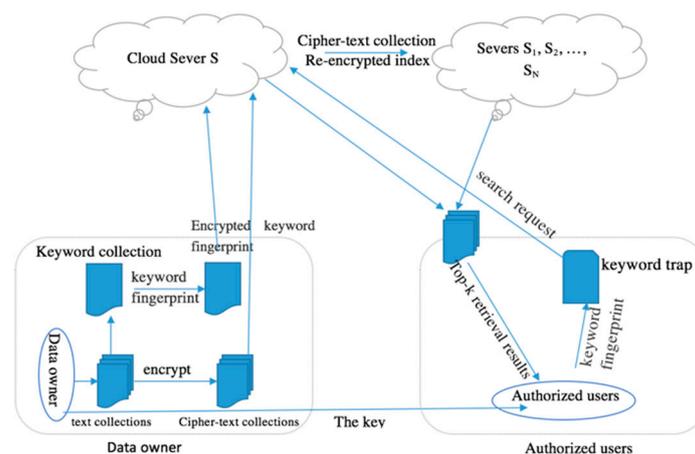


Figure 1. The system model.

Assume that the cloud servers are honest-but-curious, i.e., they will carry out search operations correctly but, nonetheless, are considered to pose threats to data security and privacy. Also assume that the number of authorized users is  $n$ , i.e.,  $U = \{u_1, u_2, \dots, u_n\}$ , and the number of files uploaded into the cloud is  $k$ , i.e.,  $F = \{f_1, f_2, \dots, f_k\}$ . To provide the functionality of protecting the security of

the data, a set of keywords  $W = \{w_1, w_2, \dots, w_p\}$  are extracted from the set of files before they are uploaded. In addition, fingerprint  $S_i$  of keyword  $w_i$  is used to construct an index  $I$  through the keyword fingerprint generation algorithm based on the MinHash technique. The set of encrypted files and the index  $I$  are eventually uploaded onto the cloud servers. The set of encrypted files are divided into  $N$  portions according to certain rules and stored on the servers  $S_1, S_2, \dots, S_N$ , respectively.

An authorized user would get the key of trapdoor provided by the data owner, generate the fingerprint of the keyword for a query and then construct the keyword trapdoor  $T_w$  before presenting it to the cloud server. To improve the accuracy of search, the cloud server needs to sort the search results. To save bandwidth, an authorized user can upload an integer  $k$  together with  $T_w$  to the cloud server for it to return the most relevant top- $k$  cipher-text files. Finally, the authorized user can use the key to decrypt the obtained cipher-text documents.

### 3.3. The Proposed Scheme

The proposed scheme focuses on the construction of encrypted search based on the MinHash algorithm that is comprised of the following steps:

- (1) Setup ( $k$ ): The data owner inputs security parameter  $k$  and outputs a  $p$ -order cyclic group  $G$  based on  $g$ . Assume that there is a one-way hash function  $F(\cdot): \{0,1\}^* \rightarrow \{0,1\}^n$ , a homomorphic hash function  $H(\cdot)$  and a hash function  $Z(\cdot)$  for compression. Let  $En(\cdot)$  be a packet encryption algorithm for file encryption,  $Index(\cdot)$  be the key index for public key encryption and  $E'_{sk}(\cdot)$  be the public key encryption algorithm for digital signature. The data owner randomly selects the master key  $s \in Z_p^*$  and an auxiliary key  $s' \in Z_p^*$ , calculates  $h = g^s$  and  $h' = g^{s-s'}$ , randomly selects a pseudo-random function  $f: \{0,1\}^k \times Z_p^* \rightarrow Z_p^*$  as well as a random parameter  $t \in \{0,1\}^k$ . The data owner could then get  $H(x): \{0,1\}^* \rightarrow Z_p^*$  and  $H'(x): \{0,1\}^* \rightarrow Z_p^*$ , select a symmetric key  $K$  for packet encryption algorithm  $En(\cdot)$  and publish  $params = (G, g, p, f, H, H', h, En(\cdot))$  to complete the generation of the public parameters and keys.
- (2) Add-user ( $u_i$ ): The data owner generates  $u_i = \{a_1, a_2, \dots, a_x\}$  according to the user's access and sends the user table to the cloud servers  $S_1, S_2, \dots, S_N$ , where  $a_i$  is the user's attribute.
- (3) The server determines whether the user has access to the file by comparing the attributes of the user and the attributes required to access the file, then sends  $(t, K, s')$  to  $u_i$ .
- (4) BuildIndexTree ( $h', W$ ): The data owner extracts the set of keywords  $W = \{w_1, w_2, \dots, w_n\}$  from the document set  $F = \{f_1, f_2, \dots, f_m\}$ , uses the keyword fingerprint generation algorithm based on MinHash, obtains the fingerprint  $S_i$  corresponding to each keyword  $w_i$ , that creates unique document identifiers for each document  $FID_j$  ( $1 \leq j \leq m$ ), where  $FID_{w_i}$  represents the set of identifiers for all documents containing the keyword  $w_i$ .
- (5) Encrypt ( $K, s', t, D, W$ ): The data owner enters the packet key  $K$ , the auxiliary key  $s'$ , the random parameter  $t$ , file  $D$  and its keyword list  $W = \{w_1, w_2, \dots, w_d\}$ , randomly selects  $r \in Z_p^*$ , calculates  $g^r, h'' = (h')^r$  and  $h^r$ , and sends  $h''$  to the cloud server  $S$ . It should then calculate  $\delta_i = f[t, H(w_i)]$  and  $E(w_i)$  according to Equation (5), make  $I = (g^r, h^r, E(w_1), E(w_2), \dots, E(w_d))$ , calculate  $C = En_K(D)$  and sends  $I$  and the encrypted index structure to the cloud server  $S$ . The set of ciphertext documents  $C = (c_1, c_2, \dots, c_m)$  are divided into  $N$  portions according to some rules and then sent to the cloud servers  $S_1, S_2, \dots, S_N$ .

$$E(w_i) = g^{r(s'+\delta_i)}, 1 \leq i \leq d \tag{5}$$

- (6) S-Encrypt( $I$ ): The server  $S$  executes the algorithm to re-encrypt the index  $I$ , enters index  $I$  and  $h''$ , calculates  $E'(w_i)$  according to Equation (6) and  $I' = (g^r, h^r, E'(w_1), E'(w_2), \dots, E'(w_d))$ , calculates  $H'[E(w_1)], H'[E(w_2)], \dots, H'[E(w_d)]$  to replace the corresponding keywords, and sends the newly generated keyword index structure to the cloud server  $S_1, S_2, \dots, S_N$ .

$$E'(w_i) = g^{r(s'+\delta_i)} \cdot h'' = (h \cdot g^{\delta_i})^r, 1 \leq i \leq d \tag{6}$$

- (7) Trapdoor ( $s', t, w_{1'}, w_{2'}, \dots, w_{w'}$ ): The authorized user enters  $s', t$  and keywords  $w_{1'}, w_{2'}, \dots, w_{w'}$  to be retrieved, randomly selects  $t'' \in Z_p^*$  and calculates  $Y = (g^r)^{t''}$ . For each keyword  $w_i$  to be searched, the user calculates the fingerprint value of the keyword  $w_i$  by using the keyword fingerprint generation algorithm MinHash ( $s', w_i$ ), which is the trapdoor  $T_i, 1 \leq i \leq w$ . The trapdoor  $T = (T_1, T_2, \dots, T_w, Y)$  is sent to the server S.
- (8) Search ( $T, I, C$ ): After the cloud server S receives the search request from the authorized user, it enters  $h''$ , calculates the Jaccard distance of the fingerprint  $S_i$  in the trapdoor T and index  $I_{w_i}$  and sends  $I''$  to servers  $S_1, S_2, \dots, S_N$ . Then, server  $S_j$  ( $j = 1, 2, \dots, N$ ) calculates  $m = A(u_i) \cap A(f_x)$  according to Jaccard similarity to query file access table of  $f_x$  and  $u_i$  in user table. If  $m \geq X_i$ ,  $f_x$  will be added to the result, otherwise  $u_i$  has no access.  $S_j$  would continue searching for files with high similarity and, when the process completes, send the accessed files to the cloud server S. All the other  $n-1$  servers repeat the same work in parallel. Server S will send the top- $t$  ( $1 \leq t \leq k$ ) files to the authorized user or "could not find relevant documents" to the user.
- (9) Decrypt ( $K, C'$ ): The authorized user inputs the packet key K and the received ciphertext  $C'$  and decrypts the top- $t$  ciphertext documents to recover the plaintext files.

### 3.4. Implementation of the Algorithm

#### 3.4.1. Keyword Fingerprint Generation Based on MinHash

The traditional MinHash algorithm can be used to generate a fingerprint for each document, making it suitable only for applicable document query scenarios. For information retrieval, the use of keywords to query documents has become more common. Therefore, the MinHash algorithm needs to be improved so that it can be used to generate more than one fingerprints for keywords. To extract more than one feature for each keyword, the following scheme makes use of the n-gram method to process keywords. In addition, since the traditional MinHash algorithm is mainly used in the plaintext data, only ordinary hash functions are used. Since our proposed scheme is to be applied to encrypted data, to improve the security of the scheme, this paper uses a keyed one-way hash function  $h(hk, \cdot)$  instead. The specific steps of generating a keyword fingerprint based on MinHash are as follows:

- (1) Input the keyword  $w$  that needs to be processed and initialize both  $\tau$ -dimensional vectors V and S to 0 in which the value of the  $\delta_i$  is the same as the number of finger of hash value generated by hash function  $h$ .
- (2) Process the keyword  $w$  by n-gram to obtain multiple features of the keyword. In this scenario, to get more accurate search results, keywords need to be divided as much as possible, i.e.,  $n = 2$ . For example, when  $w$  is encrypted, after 2-gram processing,  $\text{gramset} = \{en, nc, ry, yp, pt\}$  is obtained, where each element in  $\text{gramset}$  is a feature of the keyword  $w$ .
- (3) Use keyed one-way hash function  $h$  to calculate a hash value for each element in the  $\text{gramset}$  to protect the privacy of keywords and indexes. Hash function  $h$  can be Hmac-SHA1 or Hmac-MD5. Different hash functions will generate hash values of different sizes, which will affect the accuracy of the final search and the large the size, the more accurate the search.
- (4) Map the hash values of these elements into vector V one by one in which if the  $i$ -th bit of the hash value is 1, the  $i$ -th bit of the vector V gets incremented by 1 and if the  $i$ -th bit of the hash value is 0, the  $i$ -th bit of the vector V gets decremented by 1.
- (5) Finally, map vector V into vector S in which if the  $i$ -th bit of vector V is no less than 0, the value of the  $i$ -th bit of vector S is set to 1 and if the  $i$ -th bit of vector V is less than 0, the value of  $i$ -th bit of vector S is set to 0.
- (6) Output S as the fingerprint of the keyword.

An example of using the above algorithm is illustrated in Figure 2:

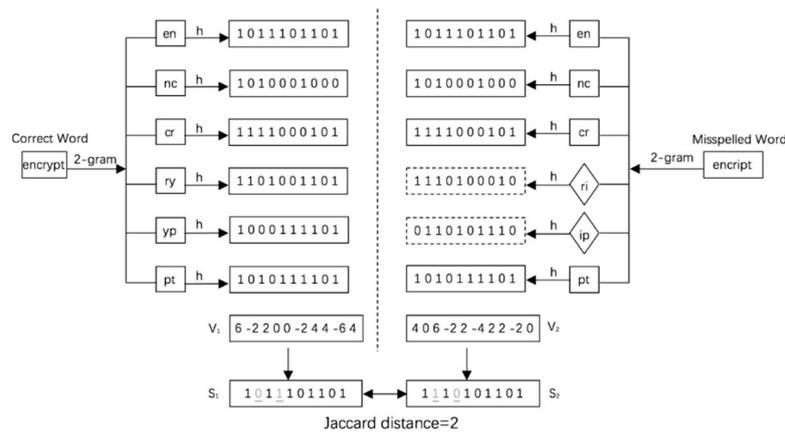


Figure 2. Key word fingerprint generation and comparison.

### 3.4.2. The Search Algorithm

Figure 2 also illustrates the principle of fuzzy search based on MinHash keyword fingerprints. Assume that the user needs to retrieve keyword  $w = \text{encrypt}$ . Due maybe to the carelessness of the user, the keyword that is actually entered is  $w' = \text{encrypt}$ , i.e.,  $y$  is misspelled as  $i$ . Using the traditional hash algorithm, any difference between two keywords will result in two completely different hash values. Using the keyword fingerprint generation algorithm, however, since the keywords have undergone  $n$ -gram processing, the value of the keyword fingerprint is determined by a plurality of elements in the gramset. When the user misspells one letter, only two out of the six elements in the gramset are changed, i.e.,  $ry$  becomes  $ri$  and  $yp$  becomes  $ip$ . Although the hash values of the two elements result in differences (dotted box in Figure 2), since the fingerprint is derived from mapping the hash values of six elements, the other four unaltered elements play a dominant role in the value of the fingerprint. Therefore, the Jaccard distance between the fingerprint of the misspelled keyword and that of the correct keyword is smaller than the fingerprint of other irrelevant keywords. Using this technique, the user can search for the correct keyword by comparing the Jaccard distance between the fingerprints even in the case of some misspellings. As can be seen in Figure 2, since the Jaccard distance between fingerprints  $S_1$  and  $S_2$  through calculation is 2, it can be determined that the keyword “encrypt” corresponding to fingerprint  $S_1$  could be a keyword that the user can use to do the search. During the search phase, the cloud server will also calculate the Jaccard distance  $J_{w_i}$  of the fingerprint trap gate  $T_w$  and the index  $S_i$  in the index  $I_{w_i} = \{S_i, FID_{w_i}\}$  according to the same principle. Therefore, according to Jaccard distance  $J_{w_i}$ , the set of top- $k$  ciphertext documents  $C' = (c_1, c_2, \dots, c_k)$  are returned to the user.

### 3.5. A New Retrieval Method

To further expand this scheme and make it suitable for large datasets, this paper proposes to construct a fingerprint index tree  $G_W$  to improve the efficiency of search, as shown in Figure 3. When an index tree is built, the root node is generated first, which is a set that is empty. Then, the fingerprint  $S_i$  of the keyword  $w_i \in W$  is calculated. If  $S_i$  is  $\tau$  bits in length,  $S_i$  can be divided into  $\tau/\lambda$  segments with each segment being expressed using  $\alpha_\rho$ , a binary bit stream of length  $\lambda$ .  $S_i$  can thus be expressed as  $\alpha_1\alpha_2 \cdots \alpha_{\tau/\lambda}$  and each  $\alpha_\rho$  represents a node. When  $\alpha_\rho$  is a leaf node,  $\{FID_{ij}, OPE(ek, Score_{ij})\}$  is inserted in the leaf node. Each path from the root node to the leaf node represents the fingerprint  $S_i$  of a keyword. The above operation is performed for each keyword until a complete fingerprint index tree is constructed.

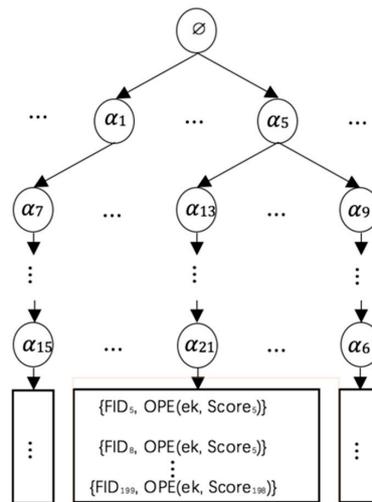


Figure 3. Illustration of the fingerprint index tree.

There are now two issues to consider. The first issue is that the simple equivalence comparison following a traditional index tree traversal method would stop as soon as the value of the compared node is different from the expected one. This obviously cannot be used for comparing Jaccard distances, neither can it be for comparing the similarity of two sets. The second issue is that if the fingerprint of a keyword and the Jaccard distance need to be calculated, traversal of the index tree needs to get to the leaf node. Therefore, if all the Jaccard distances need to be determined, traversal of the entire index tree is required, which would hardly achieve the original goal for building an index tree to improve efficiency. To achieve the goals of not necessarily traversing the whole index tree and comparing Jaccard distances, this paper proposes a new traversal method based on the characteristics of Jaccard distance as follows.

Phase 1:

Assume that the user wants to retrieve the most relevant  $k$ -documents. Before traversing the index tree, the cloud server would first construct a set  $U$  of capacity  $k$ . In the initial search phase, without any condition for decision-making, the first found leaf node can be retrieved through depth-first search and the Jaccard distance  $J_{w_i}$  between the root-to-leaf node path string  $S_i$  and the trapdoor  $T_w$  is calculated. The information associated with this leaf node  $[J_{w_i}, \{FID_{i_j}, OPE(ek, Score_{i_j})\}]$  is then stored in  $U$ . Following the depth-first search, every leaf node will be traversed, and the associated information of each leaf node is similarly stored in  $U$  in the order of  $J_{w_i}$ . When the set  $U$  is filled with  $k$  documents, search of leaf nodes stops although the documents in  $U$  at this point are not necessarily the final result of the search. The maximum value of  $J_{w_i}$  in the set  $U$  is recorded as  $J_{max}$ , and execution goes into phase 2.

Phase 2:

- (1) When an intermediate node  $S_i$  is reached during the traversal, the Jaccard distance  $\hat{J}$  between the path from the root node to  $S_i$  and  $T_w$  is compared. If  $\hat{J} > J_{max}$ , it implies that continuing the search from  $S_i$  to all the nodes beneath it will only return documents that are no more relevant than the documents in the current  $U$ . Therefore, no node after  $S_i$  will be traversed. Traversal of the index tree will continue only when  $\hat{J} \leq J_{max}$ .
- (2) If a leaf node is reached in the traversal, it must be the case that  $\hat{J} \leq J_{max}$ , meaning that the Jaccard distance calculated for the leaf node is less than or equal to the Jaccard distance  $J_{max}$  of  $U$ . If  $\hat{J} = J_{max}$ , the content of  $U$  is not changed. If  $\hat{J} < J_{max}$ , information  $[J_{w_i}, \{FID_{i_j}, OPE(ek, Score_{i_j})\}]$  associated with the leaf node has a higher correlation than information  $[J_{max}, \{FID_{i_j}, OPE(ek, Score_{i_j})\}]$

associated with  $J_{max}$  stored in the  $U$ . Then, information  $[J_{w_i}, \{FID_{i_j}, OPE(ek, Score_{i_j})\}]$  will be entered into  $U$  in an increasing order. This will result in a less relevant document to be removed from  $U$ , maintaining  $k$  documents in it. For the new set of documents in  $U$ ,  $J_{max}$  is reset, and traversal continues until all the leaf nodes are covered. It is very clear that following this method, some parts of the entire index tree will not be traversed, saving some traversal operations as well as time. The efficiency of search is therefore improved, which is very significant for massive datasets. Algorithm 1 is the pseudo-code of the search method.

---

**Algorithm 1.** *The New Method for Fingerprint Index Tree Retrieval*

---

**Input:** a: fingerprint index tree  $G_W$   
 b: search for trapdoor  $T_W$

**Output:** a sorted top- $k$  ciphertext document  $C' = (c_1, c_2, \dots, c_k)$

**Step 1:** Initialize the set  $U$  of capacity  $k$  to be null

**Step 2:** WHILE (the number of documents in  $U$  is less than  $k$ )  
     Calculate the Jaccard distance  $J_{w_i}$  between the  $S_i$  and the trapdoor  $T_W$   
     Document is inserted in  $U$  in an increasing order of the values of  $J_{w_i}$   
 END WHILE

**Step 3:** Assign the maximum value of  $J_{w_i}$  in  $U$  to  $J_{max}$

**Step 4:** FUNCTION SearchTree(a)  
     Access node a  
     B = the first child node of node a  
     WHILE (b exists)  
         Calculate the Jaccard distance  $J_{w_i}$  between the  $S_i$  and the trapdoor  $T_W$   
         Let  $\hat{J} = [J_{w_i}, \{FID_{i_j}, OPE(ek, Score_{i_j})\}]$   
         IF b has not been accessed &  $\hat{J} \leq J_{max}$   
             IF b is a leaf node &  $J_{w_i} < J_{max}$   
                 Node b is inserted in  $U$  in an increasing order of the values of  $J_{w_i}$   
                 Assign the maximum value of the  $J_{w_i}$  in  $U$  to  $J_{max}$   
             ELSE SearchTree(b)  
         END IF  
     END IF  
     B = the next child node of node a  
 END FUNCTION

**Step 5:** FOR each  $FID_{i_j} \in U$   
     Sort  $J_{w_i}$  as well as  $OPE(ek, Score_{i_j})$  from small to large  
     Get  $FID' = (FID_1, FID_2, \dots, FID_k)$   
 END FOR

**Step 6:** Find cipher document set  $C' = (c_1, c_2, \dots, c_k)$  that correspond to  $FID' = (FID_1, FID_2, \dots, FID_k)$

**Step 7:** RETURN  $C' = (c_1, c_2, \dots, c_k)$

**Step 8:** DECRYPTION  $C' = (c_1, c_2, \dots, c_k)$

---

### 3.6. Authorization and Revocation User Privileges

Before performing an encrypted search, the user needs to be authorized by the data owner. For user  $U_L$ , the data owner randomly selects a key  $SK_{U_L} \in Z_p^*$  and assigns it to the user as the query key. The user generates a re-encryption key  $RK_{U_L \rightarrow DO}$  and transmits the user ID  $U_L$  and the re-encryption key over a secret channel to the cloud server  $S$ . The authorized user  $U_L$  generates a legal query threshold based on  $SK_{U_L}$  and uses the re-encryption key  $RK_{U_L \rightarrow DO}$  to re-encrypt the query threshold to carry out the query correctly.

$$RK_{U_L \rightarrow DO} = g^{\frac{SK_{DO}}{SK_{U_L}}} \tag{7}$$

If the data owner wants to revoke the query permission of user UD, it needs to send the user's identity UD and revocation command to the cloud server S. S deletes the necessary attributes, complete legal re-encryption thresholds and query operations.

#### 4. Security and Performance Analysis

In this section, we mainly analyze the performance as well as the security of the proposed scheme.

##### 4.1. Security Analysis

**Theorem 1.** *If DDH is established, no attacker can launch keyword attacks to destroy the system.*

**Proof.** If attacker A (a cloud storage server) wins the game IND-CKA with a non-negligible probability, then challenger C can use attacker A to solve challenge DDHP with an ignorable probability. Challenger C can challenge the game in the following steps:

- System establishment: Select DDHP parameters  $h_1 = g^a, h_2 = g^b, h_3 = g^c$ , pseudo-random functions  $f: \{0,1\}^k \times Z_p^* \rightarrow Z_p^*$  and random parameters  $t \in \{0,1\}^k$ . Select hash function  $H(x): \{0,1\}^* \rightarrow Z_p^*$  and calculate  $h = h_1 = g^a$ , and  $\text{params} = (G, g, p, f, h, H)$  is used as the public parameter.
- Query before challenge: Attacker A asks challenger C for ciphertext index of keywords list  $W_i = \{w'_1, w'_2, \dots, w'_m\}$ . Challenger C sends  $W_i$  to A after encryption. For the encryption, challenger C randomly selects  $r \in Z_p^*$ , calculates  $g^r$  and  $h^r, \forall w'_j \in W_i, 1 \leq j \leq m$ , calculates  $\delta'_j = f[t, H(w'_j)], E(w'_j) = (h \cdot g^{\delta'_j})^r$ , outputs  $I = (g^r, h^r, E(w'_1), E(w'_2), \dots, E(w'_m))$  and sends I to A.
- Challenge: Attacker A randomly selects two keywords  $w'_0$  and  $w'_1$  that have not been inquired by challenger C and sends them to C. Challenger C randomly selects  $t' \in \{0,1\}$ , calculates  $\delta'_{t'} = f[t, H(w'_{t'})]$ , randomly selects  $r' \in Z_p^*$ , calculates  $E(w'_{t'}) = (h_3 \cdot h_2^{\delta'_{t'}})^{r'}$ , outputs the encrypted index  $I_{t'} = (h_2^{r'}, h_3^{r'}, E(w'_{t'}))$  of  $w'_{t'}$  and returns it to A.
- Query after challenge: Challenger C can adaptively choose the ciphertext index of query keywords for attacker A, but cannot query the ciphertext index of keywords  $w'_0$  and  $w'_1$  and the number of queries takes polynomial times of k.
- Output: Attacker A outputs conjecture  $b_A \in \{0,1\}$  on B in ciphertext index  $I_b$ . If  $b_A = b$ , i.e., the A is said to have guessed the correct answer and won the game, C answers  $c = ab$  in DDHP challenge; otherwise, if A failed, C answers  $c \neq ab$ . If  $c = ab$ , the ciphertext index  $I_{t'} = (h_2^{r'}, h_3^{r'}, (h_3 \cdot h_2^{\delta'_{t'}})^{r'}) = (g^{br'}, h^{br'}, (h \cdot g^{\delta'_{t'}})^{br'})$  of keywords  $w'_{t'}$  is a correct keyword ciphertext, here  $h = h_1 = g^a, h_3 = g^c = hb$ . If  $c \neq ab, I_{t'}$  must be the ciphertext of the correct keywords list, so there will be  $c = ab$ . If  $c \neq ab, A$  does not have a chance to win the game.

In a word, if A wins, challenger C can solve the DDHP challenge with a non-negligible probability. Therefore, if the DDH hypothesis in the group  $G_1$  is established, the scheme satisfies IND-CKA security. □

In addition, the file uses a symmetric encryption algorithm (such as DES or AES) to protect confidentiality. The keywords are encrypted through pseudo-random function  $fk$  and random parameter  $t$ . Since the attacker does not know  $t$ , it cannot know any information of the plaintext even if the keyword ciphertext is obtained, making the generated trapdoor safe. Finally, since the file index and the file itself are stored separately in servers  $S$  and  $S_1, S_2, \dots, S_N$ , as the server retrieves each keyword separately, the server does not get any more information than the retrieval results (such as which keywords are included in the ciphertext). Besides, in the algorithm  $\text{Search}(\cdot)$ , a secure multiparty computing protocol can be designed so that even if servers  $S_1, S_2, \dots, S_N$  send the cipher-text to server  $S$  after encrypting the maximum value of each obtained dataset using the protocol,  $S$  can still compare the maximum value through a secure multiparty computing protocol. However, it is still possible for

server S to know the specific values from the servers. Therefore, in the multiple server model, the data stored in the cloud is not less secure than the single server model.

#### 4.2. Performance Analysis

We performed some experiment to test the performance of the proposed scheme on some actual datasets that we obtained from <http://www/cs/cmu.edu/~enron/>. The experiment environment is as follows: the operating system is 64-bit Windows 7, the CPU is an Inter(R) Core(TM) i5 (2.8 GHz) with 4 GB of memory. We used the Java class library of PDFBox to extract the content of the PDF documents. The collection of keywords is formed after extracting the keywords of the document title and filtering out some stop words. In the experiment, Hmac-SHA1 was used as the one-way hash function with 160-bit outputs. The advantages of our proposed scheme are apparent after comparing it to the wildcard and the Gram scheme.

Figure 4 shows the comparison result on the spatial overhead, showing that with the same set of keywords, the space cost of the fingerprint is much smaller than the ciphertext fuzzy set constructed by the traditional methods. The advantage on space overhead only becomes more obvious as the size of the dataset increases.

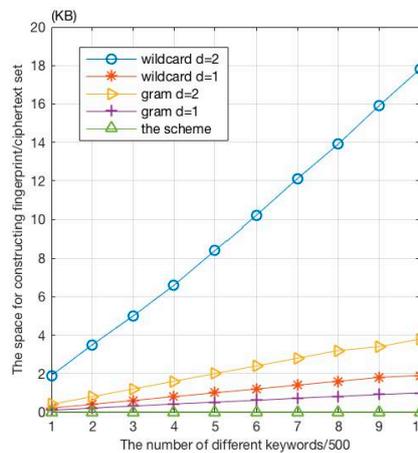


Figure 4. Spatial overhead of constructing the fuzzy set.

The time consumed for constructing the keyword fuzzy set in the MinHash fingerprint is close to the Gram method with  $d = 2$  with the time overhead remaining at a relatively low level as shown in Figure 5.

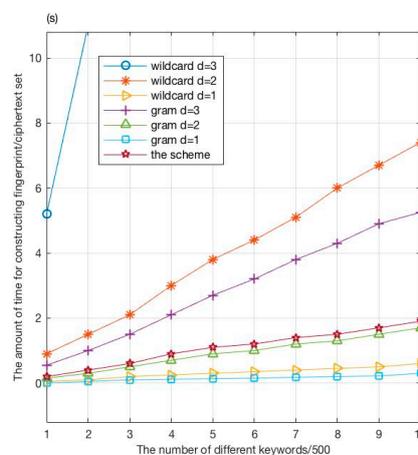


Figure 5. Time consumption for the construction of the finger/cipher-text fuzzy set.

The index construction time in Figure 6 shows that when the editing distance increases, the size of the fuzzy set increases sharply, so does the index construction time in the conventional method. When the edit distance remains the same, the fuzzy set generated by the wildcard is relatively large, so is the index construction time. Thus, the index construction time is greatly affected by the fuzzy set in the traditional method. However, the proposed scheme uses keyword fingerprints to construct a fingerprint index tree, which will not be affected by the size of the fuzzy collections. In the index construction phase, the proposed scheme calculates the key fingerprints and relevance scores, which incurs very little time overhead while reducing the storage space and optimizing the sorting results.

Figure 7 shows multi-server search efficiency, which compares the single-server model with the multi-server model where the number of servers is 1, 10, 50 and 100, respectively. As can be seen in the figure, when the number of ciphertext files remains the same, the more the number of servers participating in the search, the higher the search efficiency. Especially, as the number of files increases, the improvement on search efficiency becomes more significant. The retrieval time does not always decrease with the increase in the number of servers, however, since the server  $S$  needs to sort out the retrieval results transmitted from the servers  $S_1, S_2, \dots, S_N$  before deriving the final retrieval results. The experiment also indicates that when the number of servers is not particularly large, search for ciphertext files in the multi-server model will result in significant improvement on efficiency as the number of servers participating in the search increases. This is especially the case with a large number of documents.

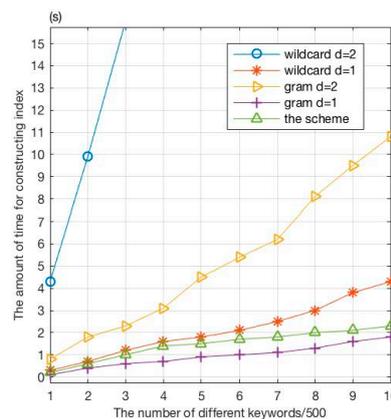


Figure 6. Time consumption for index construction.

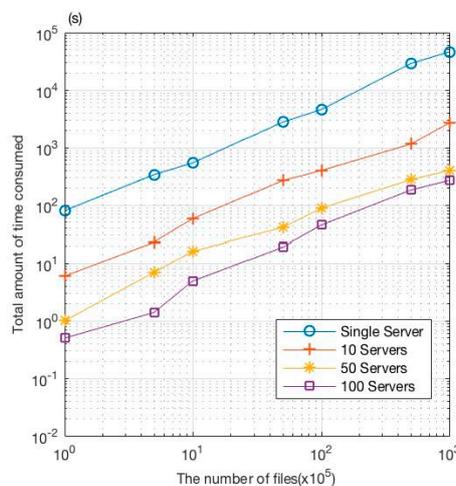


Figure 7. Efficiency of retrieval with different number of servers.

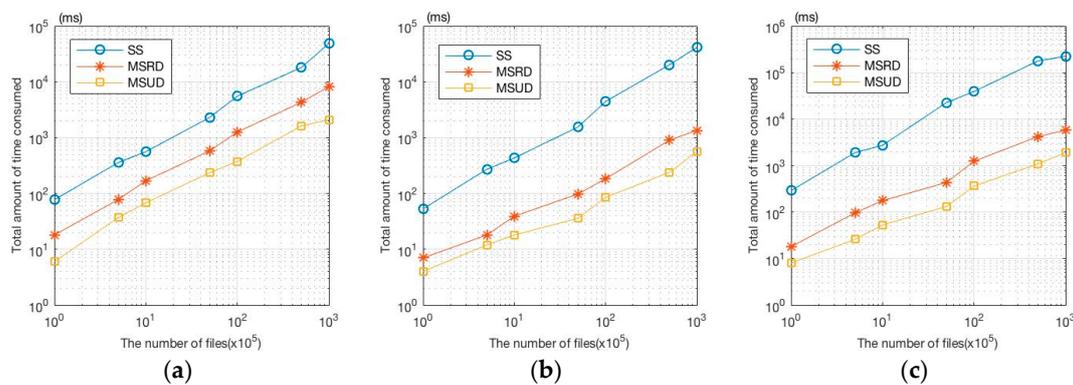
Experiment was also performed to analyze different file segmentation storage methods, i.e., random segmentation and average segmentation, under single server model and multiple server model. In the experiment, each file has 40 keywords and the keyword trapdoor that the user utilizes for retrieval contains 10 keywords. Figure 8a–c show the results with 10, 50 and 100 servers, respectively. Moreover, the number of files that are retrieved in parallel (not exceeding the trapdoor of the number of servers) may be stored randomly on the servers (MSRD) or equally divided among the servers (MSUD). Figure 8 shows the average value of 10 experiments.

### 4.3. Performance Evaluation

Performance evaluation of search results in information retrieval involves several evaluation criteria including precision and recall [24] which are defined below.

$$\text{Precision} = \frac{\text{The number of related files retrieved by the system}}{\text{Total number of files returned by the system}} \tag{8}$$

$$\text{Recall} = \frac{\text{The number of related files retrieved by the system}}{\text{Total number of relevant documents}} \tag{9}$$



**Figure 8.** Retrieval efficiency of single server/multiple servers under different segmentation storage mode. (a) 10 servers; (b) 50 servers; (c) 100 servers.

Figure 9 shows the average accuracy of multiple search results. As discussed before, different hash functions generate hash values with different numbers of bits, thus affecting the accuracy of the search results. The more the number of bits that are correct, the higher the rate of correction. From Figure 9, it can be seen that when the number of returned documents is less than 60, Hmac-SHA1 should be used and the rate of correction of the results is close to 100%. As the number of returned documents increases further, some of the lower-ranked documents will be returned some of which may be irrelevant documents, causing the accuracy of search results to follow a downward trend. Another solution is to use Hmac-MD5 as the hash function. However, since the number of bits in Hmac-MD5 (128 bits) is less than that in Hmac-SHA1 (160 bits), Hmac-MD5 makes the generated fingerprints incapable of better reflecting keyword characteristics. Therefore, the accuracy of the search results when using Hmac-MD5 is lower than that using Hmac-SHA1. In both cases, however, as the number of irrelevant documents returned increases, the accuracy rate will follow a downward trend. This experiment demonstrates that the number of bits generated by the hash function used in the proposed scheme will have an impact on the correction rate of the search results.

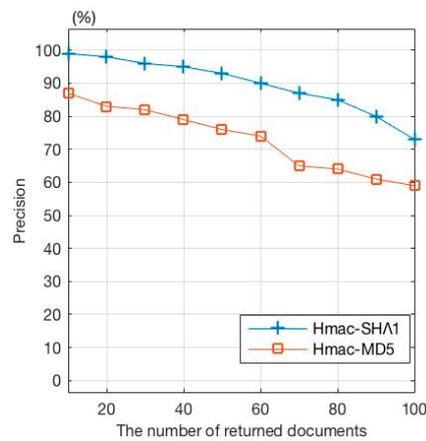


Figure 9. Evaluation of precision of search results.

Figure 10 shows the average rate of recall for multiple search results from which we can see that in the proposed scheme that uses Hmac-SHA1 as the hash function, the recall rate rises as the number of returned documents increases. Because it is the ratio of related documents that are returned to all relevant documents, the recall rate is lower when the number of returned documents is small. Since the ranking algorithm in the proposed scheme can optimize the sorted results, when the number of returned documents reaches 90, the recall rate approaches 100%. The use of Hmac-MD5 results in a lower recall rate as expected for a similar reason. That is, since the number of bits (128 bits) that it generates as the message digest is less than that generated by Hmac-SHA1 (160 bits), the resulting fingerprints generated cannot better distinguish between the characteristics of keywords. As the number of returned documents increases, the recall rate will gradually go up after more related documents are returned. The experiment proves that the number of bits generated by the hash function used to generate message digests definitely has an impact on the recall rate of search results.

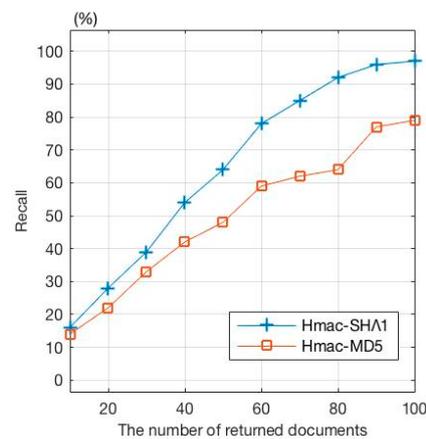


Figure 10. Evaluation of recall rate of search results.

### 5. Conclusions

With the rapid development and maturity of cloud computing, more and more users will choose to transfer data to cloud servers for storage in order to save storage as well as maintenance costs. Thus, security of stored data and privacy of user search has become a great challenge. Searchable encryption technology is a viable solution that would protect user data in the cloud and allow the user to access encrypted data on the cloud servers. Fuzzy keyword searchable encryption represents further advancement of the searchable encryption technology, for it can tolerate minor spelling errors and inconsistencies in search requests.

This paper proposed an efficient fuzzy keyword search scheme for multi-server and multi-user environments. Major advantages of the proposed scheme include reduced overhead of keyword index space storage through generating the keyword fingerprint, improved efficiency and accuracy of retrieval and provable security of keyword trapdoor. Analysis and experimental evaluation using the open Enron dataset demonstrated that the scheme proposed in this paper can effectively boost the usability of systems and the efficiency of document retrieval. In the future, we plan to expand the attack model to allow attackers to assume a variety of roles, such as registered user, registered server, network eavesdropper, etc. as the basis to further improve the security and the usability of the proposed scheme.

**Author Contributions:** Jingsha He, Jianan Wu and Nafei Zhu conceived and developed the proposed scheme; Jianan Wu performed the experiments and analyzed the results along with Nafei Zhu; Nafei Zhu coordinated the preparation and revision of the manuscript; Muhammad Salman Pathan helped in reviewing and editing the manuscript. Each author has made his/her full effort in completing the work reported in this paper.

**Acknowledgments:** The work in this paper has been supported by National Natural Science Foundation of China (No. 61602456) and National High Technology Research and Development Program of China (863 Program) (No. 2015AA017204).

**Conflicts of Interest:** The authors declare no conflict of interest.

## References

1. Song, D.X.; Wagner, D.; Perrig, A. Practical techniques for searches on encrypted data. In Proceedings of the IEEE Symposium on Security and Privacy, Berkeley, CA, USA, 14–17 May 2000; pp. 44–55.
2. Chang, Y.C.; Mitzenmacher, M. Privacy preserving keyword search on remote encrypted data. In Proceedings of the International Conference on Applied Cryptography and Network Security, New York, NY, USA, 7–10 June 2005; pp. 442–445.
3. Wang, C.; Cao, N.; Ren, K.; Lou, W. Enabling secure and efficient ranked keyword search over outsourced cloud data. *IEEE Trans. Parallel Distrib. Syst.* **2012**, *23*, 1467–1479. [[CrossRef](#)]
4. Cao, N.; Wang, C.; Li, M.; Ren, K.; Lou, K. Privacy-preserving multi-keyword ranked search over encrypted cloud data. *IEEE Trans. Parallel Distrib. Syst.* **2014**, *25*, 222–233. [[CrossRef](#)]
5. Li, J.; Wang, Q.; Wang, C.; Cao, N.; Ren, K.; Lou, K. Fuzzy keyword search over encrypted data in cloud computing. In Proceedings of the IEEE INFOCOM, San Diego, CA, USA, 14–19 March 2010; pp. 441–445.
6. Suresh, K.B. Towards an effective fuzzy keyword search and ranking framework for file information management system. *Int. J. Comput. Sci. Technol.* **2012**, *3*, 556–559.
7. Liu, C.; Zhu, L.; Li, L.; Tan, Y. Fuzzy keyword search on encrypted cloud storage data with small index. In Proceedings of the 2011 IEEE International Conference on Cloud Computing and Intelligence Systems, Beijing, China, 15–17 September 2011; pp. 269–273.
8. Wang, B.; Yu, S.; Lou, W.; Hou, Y.T. Privacy-preserving multi-keyword fuzzy search over encrypted data in the cloud. In Proceedings of the IEEE INFOCOM 27, Toronto, ON, Canada, 27 April–2 May 2014; pp. 2112–2120.
9. Zirtol, K.A.; Noroozi, M.; Eslami, Z. Multi-user searchable encryption scheme with general access structure. In Proceedings of the 2015 2nd International Conference on Knowledge-Based Engineering and Innovation, Tehran, Iran, 5–6 November 2015; pp. 399–404.
10. Li, J.; Chen, X.; Liu, Z.; Jia, C. Privacy-preserving data utilization in hybrid clouds. *Future Gener. Comput. Syst.* **2014**, *30*, 98–106. [[CrossRef](#)]
11. Goh, E.J. Secure indexes. IACRCryptology ePrint Archive, Submission, 2003.
12. Chang, C.; Zhang, L. An efficient service discovery algorithm for counting Bloom filter-based service registry. In Proceedings of the 2009 IEEE International Conference on Web Services, Los Angeles, CA, USA, 6–10 July 2009; pp. 157–164.
13. Curtmola, R.; Garay, J.; Kamara, S.; Ostrovsky, R. Searchable symmetric encryption: Improved definitions and efficient constructions. *J. Comput. Secur.* **2011**, *19*, 895–934. [[CrossRef](#)]
14. Broder, A. On the resemblance and containment of documents. In Proceedings of the International Conference on Compression and Complexity of Sequences, Positano, Italy, 11–13 June 1997; pp. 21–29.

15. Agrawal, R.; Kiernan, J.; Srikant, R.; Xu, Y. Order preserving encryption for numeric data. In Proceedings of the 23rd ACM SIGMOD International Conference on Management of Data, Paris, France, 13–18 June 2004; pp. 563–574.
16. Boneh, D.; Waters, B. Conjunctive, subset, and range queries on encrypted data. In Proceedings of the Theory of Cryptography Conference, Amsterdam, The Netherlands, 21–24 February 2007; pp. 535–554.
17. Chai, Q.; Gong, G. Verifiable symmetric searchable encryption for semi-honest-but-curious cloud servers. In Proceedings of the 2012 IEEE International Conference on Communications, Ottawa, ON, Canada, 10–15 June 2012; pp. 917–922.
18. Yang, Y.; Lu, H.; Weng, J. Multi-user private keyword search for cloud computing. In Proceedings of the IEEE 3rd International Conference on Cloud Computing Technology and Science, Athens, Greece, 29 November–1 December 2011; pp. 264–271.
19. Bijral, S.; Mukhopadhyay, D. Efficient fuzzy search engine with B-tree search mechanism. In Proceedings of the 2014 International Conference on Information Technology (ICIT), Bhubaneswar, India, 22–24 December 2014; pp. 1–5.
20. Wang, C.; Cao, N.; Li, J. Secure ranked keyword search over encrypted cloud data. In Proceedings of the IEEE International Conference on Distributed Computing Systems (ICDCS), Genova, Italy, 21–25 June 2010; pp. 253–262.
21. Pan, J.; Manocha, D. Bi-level locality sensitive hashing for k-nearest neighbor computation. In Proceedings of the 2012 IEEE 28th International Conference on Data Engineering, Washington, DC, USA, 1–5 April 2012; pp. 378–389.
22. Lu, I.; Nikova, S.; Hartel, P.; Jonker, W. Public-key encryption with delegated search. In Proceedings of the 9th International Conference on Applied Cryptography and Network Security, Nerja, Spain, 7–10 June 2011; Lopez, J., Tsudik, G., Eds.; Springer: Berlin/Heidelberg, Germany, 2011; pp. 532–549.
23. Fu, Z.; Hung, F.; Sun, X.; Vasilakos, A.; Yang, C.N. Enabling semantic search based on conceptual graphs over encrypted outsourced data. *IEEE Trans. Serv. Comput.* **2016**. [[CrossRef](#)]
24. Shekhar, N.; Sampat, K.; Chandawalla, C.; Shah, J. Implementation of fuzzy keyword search over encrypted data in cloud computing. *Procedia Comput. Sci.* **2015**, *45*, 499–505. [[CrossRef](#)]



© 2018 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<http://creativecommons.org/licenses/by/4.0/>).