Secure Multi-keyword Fuzzy Search Supporting Logic Query over Encrypted Cloud Data

Qi Zhang[1], Shaojing Fu[1,2](✉), Nan Jia[1], Jianchao Tang[1], and Ming Xu[1]

[1] College of Computer, National University of Defense Technology, Changsha, China
shaojing1984@163.com

[2] State Key Laboratory of Cryptology, Beijing, China

Abstract. Compared with exact search, fuzzy search will meet more practical requirements in searchable encryption since it can handle spelling errors or search the keywords with similar spelling. However, most of the existing fuzzy search schemes adopt bloom filter and locality sensitive hashing which cannot resist Sparse Non-negative Matrix Factorization based attack (SNMF attack). In this paper, we propose a new secure multi-keyword fuzzy search scheme for encrypted cloud data, our scheme leverages random redundancy method to handle the deterministic of bloom filter to resist SNMF attack. The scheme allows users to conduct complicated fuzzy search with logic operations ("AND", "OR" and "NOT"), which can meet more flexible and fine-grained query demands. The theoretical analysis and experiments on real-world data show the security and high performance of our scheme.

Keywords: Searchable encryption · Fuzzy search · Logic query · Bloom filter

1 Introduction

Recently, Cloud storage services have become more and more prevalent and many individuals and businesses choose to outsource their local data to remote cloud service providers to reduce local storage and computing overhead. In order to protect the privacy of the outsourced data, data encryption is usually used, making it impossible for an attacker to recover the original data from the encrypted data. However, data encryption will cause a decrease in data availability, making it difficult to perform operations such as information retrieval on ciphertext. How to implement secure search over encrypted cloud data becomes a topic worth studying.

Searchable encryption technology can realized the function of information retrieval on encrypted cloud data while protecting privacy. Most of the searchable encryption schemes can only support exact keyword search which does not have fault tolerance. Fuzzy search is mainly designed for misspelling, similar

J. Li et al. (Eds.): SPNCE 2019, LNICST 284, pp. 210–225, 2019.
https://doi.org/10.1007/978-3-030-21373-2_17

query and other scenarios. According to some similarity metrics, such as edit distance, Jaccard distance, Euclidean distance etc., the similarity between the index and the trapdoor can be calculated, and the scheme can return the results with higher similarity to the query. Existing fuzzy searches mainly fall into two directions. One is to construct a pre-defined wildcard-based fuzzy keyword set for each keyword which will result in high storage and can only support single keyword search. The other is using bloom filter and locality sensitive hashing [8] to construct index which can map similar keywords to the same position to realize multi-keyword fuzzy search. However, the structure of bloom filter cannot resist ciphertext-only attack which will leak the search pattern [12].

Aiming at the security flaws of existing fuzzy search schemes, our paper proposes a multi-keyword fuzzy search scheme with high security and can support logical query. Our paper add random number redundancy to confuse the original keyword features, so that the search pattern will be protected and the scheme can resist ciphertext-only attack. In addition, our paper extends to implement logic query function, i.e. "AND", "OR" and "NOT" which is practical in applications. Users can customize the files that must contain certain keywords, or the files that does not contain certain keywords, thereby obtaining the search results that are more in line with the user's needs. Finally, this paper theoretically analyzes the security of the scheme and analyzes its performance on the real-world text dataset. Our contributions can be summarized as follows:

(1) To resist the ciphertext-only attack basing on sparse non-negative matrix factorization, we randomized the index by adding random redundancy to mask the deterministic of bloom filter and improve the security of scheme proposed by Wang et al. [15] as a result.
(2) To realize flexible query for users, we extend the scheme to support logic query, the mixed "AND", "OR" and "NOT" operations, to exclude the results users don't need, and pick out the results they want, and last rank the relevance scores according to the possibly existing keywords.
(3) The scheme is proved secure against two threat model by theoretical analysis, and has high performance by evaluating on the real-world dataset.

The remaining sections of this paper are organized as follows. In Sect. 2, we outline the system model, threat model, design goals and the notations used in this scheme. And Sect. 3 introduces the basic theoretical knowledge of the scheme. Section 4 describes the technical details of the scheme. Section 5 gives the theoretical analysis of the security of the scheme. The experiment results of the scheme are given in Sect. 6. Section 6.1 gives a brief introduction of the related work. Finally Sect. 7 concludes the paper.

2 Problem Formulation

2.1 System Model and Threat Model

As shown in Fig. 1, the system consists of three entities: data owner, cloud server and data user. Before outsourcing data to the server, the owner needs to encrypt

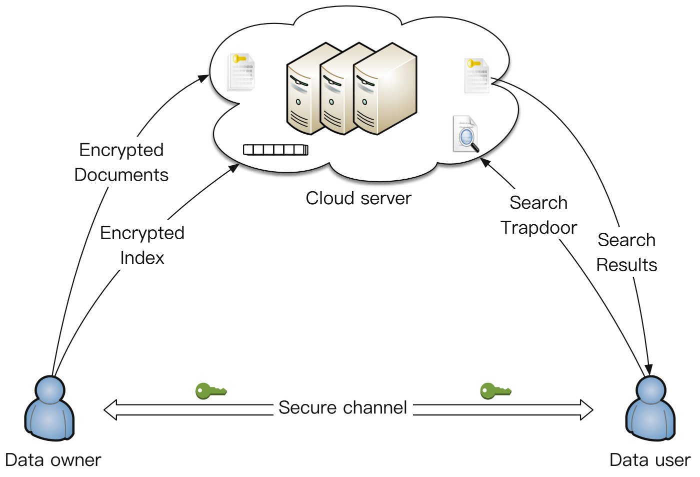


Fig. 1. System model

the data, construct secure indexes, and upload them to the cloud. The keys used in symmetric encryption algorithm for documents and secure kNN algorithm for indexes are transmitted to the data user via a secure channel. When searching a certain file, data user generates the trapdoor according to the input of "AND", "OR", "NOT" operations, and encrypts it using the same key for index encryption, and then sends it to cloud server. After receiving the query trapdoor, cloud server calculates the relevance scores one by one for the encrypted indexes, and returns the most satisfactory results to data user. When the query results returned, data user decrypts them locally using the key for document encryption.

We assumes that the cloud server is honest but curious. In other words, the cloud server will honestly follow the steps of algorithms, but will be curious about the content of files, keywords, and other additional information. In this model, in addition to the encrypted information introduced above, cloud server will obtain additional background information, such as trapdoor correlation. This information will be used for statistical attacks to infer the keywords contained in search requests.

2.2 Preliminaries

Bloom Filter. Bloom Filter [1] is a data structure with high space efficiency which can determine whether the collection contain the element.

The structure of bloom filter is shown in Fig. 2. Bloom filter uses a fixed-length vector with m bits to represent a set of elements. For a given collection with n elements $S = \{s_1, s_2, \ldots, s_n\}$, use l hash functions from a hash family $H = \{h_i | h_i : S \to [1, m], 1 \le i \le l\}$ to map the elements to l positions of the vector and set the value to be 1, others set to be 0. The l position represents the element. To check whether the input element x is in the collection, first calculate the element x with the same l hash function to get l positions. If the value in all positions are 0, then $x \notin S$; otherwise, we predicate $x \in S$.

Bloom filter have significant advantages. First of all, the size of the bloom filter is fixed and is not limited by the number of elements in the set. At the same time, it does not need to store the information of the element itself, which will not leak any information of the collection and elements. But on the other hand, it has false positive result, that is, the non-existent element will be predicated to existent. The false positive rate is approximately $(1 - e^{-\frac{ln}{m}})^l$.

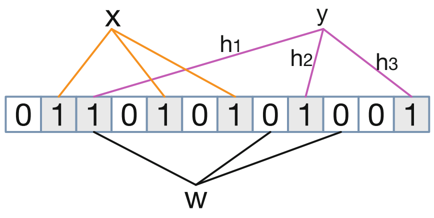


Fig. 2. Structure of bloom filter

Locality Sensitive Hashing. Locality sensitive hashing function is an algorithm used to solve near-neighbor search. It can map similar elements to the same bucket with high probability, thus deciding the similarity between elements. The hash function family is defined as [8]:

Definition 1. *For any two points x and y are satisfied:*
If $d(x, y) \le r_1$, $Pr[h(x) = h(y)] \ge p_1$;
If $d(x, y) \ge r_2$, $Pr[h(x) = h(y)] \le p_2$.
Where $d(x, y)$ is the distance between x and y, $r_1 < r_2$, $p_1 > p_2$, and $h(x)$ is the hash value for x. Then the hash function family H can be defined as (r_1, r_2, p_1, p_2)-sensitive.

This definition shows that two adjacent points in the original space will be mapped into two points still adjacent, and the non-adjacent points are still not adjacent after mapping. Based on this property, LSH can be used in bloom filter to replace the original hash function to achieve fuzzy search.

3 The Proposed Scheme

3.1 Previous Scheme and Security Defect

The first multi-keyword fuzzy search scheme using bloom filter and locality sensitive hashing is proposed by Wang et al. [15]. There are 4 main processes in the scheme:

1. Keyword transformation. Since LSH function uses a vector as input, this process converts the keyword string into a vector. The scheme uses bi-gram set that contains all of the two consecutive letters to represent the keyword. For example, the bi-gram set of the keyword "network" is {ne, et, tw, wo, or, rk}. Later, the scheme transforms the bi-gram set to 26^2-bit vector, and we eet the position to 1 if the element exist in the bi-gram set. Through this expression, a keyword can have many different forms of spelling errors, but it can still be represented by a vector with very close distance with the correct one.
2. Index construction. This process use bloom filter with LSH to build an index for each file, which will hash similar inputs to the same output with high probability.
3. Trapdoor generation. Generate trapdoor using the same process as index construction.
4. Search. This process calculates the relevance score to reflect the relevance of multiple keywords and documents, and rank it according to the result of inner product.

Ciphertext-Only Attack. The main process in Wang's scheme can be simply expressed as the following function:

$$\begin{aligned} I_i &= f(LSH(P_i), k_i) \\ T_j &= f(LSH(Q_j), k_i) \\ R &= I_{enc,i}^T T_{enc,j} = I_i^T T_j \end{aligned} \tag{1}$$

Where k_i is the key for pseudo-random function f, f can be equivalent to position permutation, P_i and Q_j are plaintext keyword vectors, I_i and T_j are m-bit bloom filter obtained by P_i and Q_j through LSH and pseudo-random function, $I_{enc,i}$ and $T_{enc,j}$ are encrypted from I_i and T_j, R is the relevance score.

Analyzing above equations, I_i and T_j cannot be inferred from P_i and Q_j without knowing the key K. However, the generation of the bloom filter is deterministic which means same query vector will generate the same bloom filter and leak the search pattern as a result. According to the analysis by Lin et al. [12], an adversary can conjecture I_i and T_j by deploying sparse non-negative matrix factorization (sparse-NMF) algorithm [9] on the encrypted index $I_{enc,i}$ and trapdoor T_{enc}. In other words, The basic scheme is vulnerable to the ciphertext-only attack (COA). Although I_i and T_j do not directly leak the plaintext P_i and Q_j, similar I_i and T_j will reflects the relationship of plaintext due to the deterministic of LSH and f and make statistical analysis attack feasible which will reveal the frequency and other information of keywords.

3.2 Secure Scheme Supporting Logic Query

To resist above-mentioned ciphertext-only attack (COA) against the previous scheme, we extend m-bit vector of the original scheme to $(m+U+1)$-bit, which is a combination of bloom filter and random numbers. With the confusion of random numbers, the location of the keywords and random numbers is indistinguishable. Even if two identical trapdoor, it is impossible to determine whether the same position corresponds to the keyword or the random number, and thus it is impossible to determine whether the query is the same. It is also unable to perform statistical analysis to obtain keyword information such as frequency.

Then to increase flexibility of query, we also design a scheme to support logic query. Users can input keywords with customization that must exist, must not exist and possibly exist. So that user can exclude undesirable documents, and select requisite documents.

First, the TF-IDF algorithm used in this scheme is defined as:

$$\begin{aligned} S &= \sum_{w_i \in Q} TF_{f_i,w_i} \times IDF_{w_i} \\ &= \sum_{w_i \in Q} \frac{ln(1+N_{f,w_i})}{\sqrt{\sum_{w_i \in W} (ln(1+N_{f,w_i}))^2}} \times \frac{ln(1+N/N_{w_i})}{\sqrt{\sum_{w_i \in W} (ln(1+N/N_{w_i}))^2}} \end{aligned} \tag{2}$$

Then the details of enhanced scheme is shown in the following aspects:

- $\{SK, sk\} \leftarrow KeyGen(1^\rho)$. This algorithm is executed on data owner side. Given a parameter, the algorithm will generate the key $SK = \{S, M_1, M_2\}$ for the index and trapdoor encryption. Our scheme extends the vector from m bits to $(m+U+1)$ bits, where m-bit vector represents the bloom filter and U-bit is for random numbers. Thus S is a $(m+U+1)$-bit vector $S \in \{0,1\}^{m+U+1}$, $\{M_1, M_2\}$ are two $(m+U+1) \times (m+U+1)$-bit invertible matrices. At the same time, the key sk of the symmetric encryption algorithm for file encryption and decryption will be generated.
- $C \leftarrow Enc(F, sk)$: Data owner encrypts the plaintext document collection F using a symmetric encryption algorithm, like AES. And the encrypted documents will be finally uploaded to the cloud server for storage.
- $BF \leftarrow BuildBF(W)$: This algorithm maps the input keyword set to the bloom filter to generate an index. First initialize a m-bit bloom filter BF, each of which is set to 0. Then, transform each keyword to a 26^2-bit vector, $W = \{w_1, w_2, \cdots, w_n\}, w_i \in \{0,1\}^{26^2}$. Then select l independent hash function

h_j from p-stable LSH function family $H = \{h : \{0,1\}^{26^2} \to \{0,1\}^m\}$ and combine l hash function with the pseudo-random function f_{k_i} together to generate a new hash function $\{g_i | g_i = f_{k_i} \circ h_i, h_i \in H, 1 \leq i \leq l\}$. Those pseudo-random functions is equivalent to randomly permutate the positions of bloom filter, and k_i is the key of the random function, which eliminates the connection between the keyword and bloom filter to resist known background attack [15]. Then, for each keyword vector w_i, use l hash functions g_i to get l positions, set the corresponding position of the bloom filter to its weight. In this case, the bloom filter is constructed.

- $I_{enc} \leftarrow Index(F, SK)$: This algorithm is used to construct the secure index vector. Data owner extracts the keywords of each document f_i in the document collection F, and obtains the keyword set W_{f_i}. Then call the $BuildBF(W_{f_i})$ algorithm to generate a m-bit bloom-filter-based index vector I_i for each file whose value is set to TF_{w_i}. If the collision occurs, the maximum value is retained. Then, each vector is expanded to a $(m + U + 1)$ bit vector. The $(m + j)^{th}$ $(j \in [1, U])$ bit is set to a random number $\varepsilon^{(j)}$. The $(m + U + 1)^{th}$ bit is set to -1. The maximum value D of the vector will be sent to the user after expanding. Next use the secure kNN algorithm [17] to encrypt the index vector. First split the index vector I_i into two random vectors according to the vector S, namely $\{I_i^{'}, I_i^{''}\}$, S is the secret vector used for splitting. When $S[j] = 0$, set $I_i^{'}[j] = I_i^{''}[j] = I_i[j]$; when $S[j] = 1$, set $I_i^{'}[j]$ and $I_i^{''}[j]$ as random numbers, and $I_i^{'}[j] + I_i^{''}[j] = I_i[j]$. Finally, each index vector is encrypted as $I_{enc,i} = \{M_1^T I_i^{'}, M_2^T I_i^{''}\}$.
- $T_{enc} \leftarrow Trapdoor(Q, SK)$: We select v random positions $j (j \in [m, m+U])$ to be the random number $\sigma^{(j)}$ which can be equivalent to the keyword of "OR" operation. Then to realize logic query, the query keywords Q consisting of t keywords are arranged in the order of "OR", "AND" and "NOT" operation, defined as $(a_1, a_2, \cdots, a_{l_1}, a_{l_1+1}, \cdots, a_{l_1+v})$, $(a_{l_1+v+1}, a_{l_1+v+2}, \cdots, a_{l_1+v+l_2})$, $(a_{l_1+v+l_2+1}, a_{l_1+v+l_2+2}, \cdots, a_N)$, which contains l_1 "OR" operation keywords, v random numbers, l_2 "AND" operation keywords, and $N - l_1 - l_2 - v$ "NOT" operation keywords, a_i is the value of the keyword w_i. For l_1 keywords of "OR" operation, the value is set as its own IDF value. For other keywords, assign a random value satisfying $\sum_{i=1}^{j-1} l \cdot a_i \cdot D < a_j (j = l_1 + 1, l_1 + 2, \cdots, N)$. Then call $BuildBF(Q)$ for the query keyword to generate a m-bit bloom-filter-based query vector T. For collision, the maximum value will be selected as the weight. Append the v positions to expand T to $(m + U + 1)$-bit vector and set the $(m + U + 1)^{th}$ bit to $t = a_{l_1+v+l_2+1}$. Next, split T by secret vector S into two random vectors $\{T^{'}, T^{''}\}$, if $S[j] = 1$, set $T^{'}[j] = T^{''}[j] = T[j]$; if $S[j] = 0$, $T^{'}[j]$ and $T^{''}[j]$ are set to random numbers, and $T^{'}[j] + T^{''}[j] = T[j]$. Finally, the trapdoor is encrypted as $T_{enc} = \{M_1^{-1} T^{'}, M_2^{-1} T^{''}\}$.
- $R \leftarrow Search(I_{enc,i}, T_{enc})$: After receiving the trapdoor T_{enc} uploaded by the data user, cloud server calculates the relevance score between T_{enc} and the index vector stored on the cloud server to get the most relevant results.

The relevance score is calculated as:

$$\begin{aligned}
S &= I_{enc,i} \cdot T_{enc} \\
&= (M_1^T I_i^{'}) \cdot (M_1^{-1} T^{'}) + (M_2^T I_i^{''}) \cdot (M_2^{-1} T^{''}) \\
&= I_i \cdot T \\
&= r \cdot (P \cdot Q - t) \\
&= r \cdot (\sum_{i=1}^{N} \gamma_i \cdot score(w_i, F_j) \cdot a_i - t) \\
&= r \cdot (\sum_{i=1}^{l_1+v} \gamma_i \cdot score(w_i, F_j) \cdot a_i + \sum_{i=l_1+v+1}^{l_1+v+l_2} \gamma_i \cdot score(w_i, F_j) \cdot a_i \\
&\quad + \sum_{i=l_1+v+l_2+1}^{N} \gamma_i \cdot score(w_i, F_j) \cdot a_i - t)
\end{aligned} \tag{3}$$

(1) "NOT" operation. Since $t = a_{l_1+v+l_2+1} > \sum_{i=1}^{l_1+v+l_2} \gamma_i \cdot score(w_i, F_j) \cdot D$, where $\gamma_i (0 \le \gamma_i \le l)$ is the amount of the keyword exist in bloom filter after collision. Once there exist "NOT" operation keywords, $R_j = r \cdot (\sum_{i=1}^{l_1+v+l_2} \gamma_i \cdot score(w_i, F_j) \cdot a_i + \sum_{i=l_1+v+l_2+1}^{N} \gamma_i \cdot score(w_i, F_j) \cdot a_i - a_{l_1+v+l_2+1}) > 0$, otherwise $R_j < 0$. This process can eliminate results that do not meet the requirements, and only go to the next process when $R_j < 0$.

(2) "AND" operation. Mod R_j by $(-r \cdot a_{l_1+v+l_2+1}, r \cdot \gamma_{l_1+v+l_2} \cdot a_{l_1+v+l_2}, \cdots, r \cdot \gamma_{l_1+v+1} \cdot a_{l_1+v+1})$ iteratively and judge whether the keyword is exist. First, mod R_j with $-r \cdot a_{l_1+v+l_2+1}$ to eliminate the effect of s and obtain the remainder result.

$$\begin{aligned}
R_j &= (r \cdot (\sum_{i=1}^{l_1+v} \gamma_i \cdot score(w_i, F_j) \cdot a_i + \sum_{i=l_1+v+1}^{l_1+v+l_2} \gamma_i \cdot score(w_i, F_j) \cdot a_i \\
&\quad + \sum_{i=l_1+v+l_2+1}^{N} \gamma_i \cdot score(w_i, F_j) \cdot a_i) - t) mod(-r \cdot a_{l_1+v+l_2+1}) \\
&= r \cdot (\sum_{i=1}^{l_1+v} \gamma_i \cdot score(w_i, F_j) \cdot a_i + \sum_{i=l_1+v+1}^{l_1+v+l_2} \gamma_i \cdot score(w_i, F_j) \cdot a_i \\
&\quad - a_{l_1+v+l_2+1}) mod(-r \cdot a_{l_1+v+l_2+1}) \\
&= r \cdot (\sum_{i=1}^{l_1+v} \gamma_i \cdot score(w_i, F_j) \cdot a_i + \sum_{i=l_1+v+1}^{l_1+v+l_2} \gamma_i \cdot score(w_i, F_j) \cdot a_i)
\end{aligned} \tag{4}$$

The quotient of equation is 1, and the remainder is the sum of the previous $l_1 + v + l_2$ items.

Then we mod R_j with the remaining reverse ordering keywords $(r \cdot \gamma_{l_1+v+l_2} \cdot a_{l_1+v+l_2}, r \cdot \gamma_{l_1+v+l_2-1} \cdot a_{l_1+v+l_2-1}, \cdots, r \cdot \gamma_{l_1+v+1} \cdot a_{l_1+v+1})$ in turn. If the quotient of $R_j mod(r \cdot \gamma_{l_1+v+l_2} \cdot a_{l_1+v+l_2})$ is greater than 1, it can be determined that the keyword $w_{l_1+l_2}$ exists in the index vector, then assign the remainder to $R_j = r \cdot (\sum_{i=1}^{l_1+v} \gamma_i \cdot score(w_i, F_j) \cdot a_i + \sum_{i=l_1+v+1}^{l_1+v+l_2-1} \gamma_i \cdot score(w_i, F_j) \cdot a_i)$. Otherwise we break the iteration and check next index.

(3) "OR" operation. When all the keywords in "AND" operation are successfully checked, the obtained $R_j = r \cdot (\sum_{i=1}^{l_1+v} \gamma_i \cdot score(w_i, F_j) \cdot a_i)$ is the final relevance score for rank search.

- $PR \leftarrow Dec(R, sk)$. The data user decrypts the returned result R using the key sk transmitted from the data owner via the secure channel, resulting in the plaintext results PR.

4 Security Analysis

4.1 Known Ciphertext Model

The cloud server can only obtain encrypted documents, encrypted indexes and trapdoors under this model. The difference between the indexes and documents depends mainly on the index generation algorithm $I \leftarrow Index(F, SK)$ and file encryption algorithm $C \leftarrow Enc(F, sk)$. The index vector has $(m + U + 1)$ bits. The first m bits represent the weight of the keyword. The U bits are randomly selected. And the last 1 bit is set to -1.

For the index generation, the index vector is first split into two vectors. The value of the vector is randomly set if the value in S is 1. Assuming that the total number of "1" in the first m bit and the last one bit is μ_1, and each dimension of index is η_f bits, then there will be $(2^{\eta_f})^{\mu_1} \cdot (2^{\eta_f})^U$ possible values. Then the two vectors are encrypted by two random $(m + U + 1) \times (m + U + 1)$-bit secret matrices. Assuming each element in the matrix has η_M bits, then there will be $(2^{\eta_M})^{(m+U+1)^2 \times 2}$ possible values. Therefore, the probability of same indexes for two documents can be calculated as:

$$\begin{aligned} P_d &= \frac{1}{(2^{\eta_f})^{\mu_1} \cdot (2^{\eta_f})^U \cdot (2^{\eta_M})^{(n+U+1)^2 \times 2}} \\ &= \frac{1}{2^{\mu_1 \eta_f + U\eta_f + 2\eta_M (n+U+1)^2}} \end{aligned} \tag{5}$$

In addition, under known ciphertext attack, it is also necessary to consider the case where multiple ciphertext pairs are known to be $(I_i^{'}, T^{'})$ and the relevance score for each result after querying Security. According to the attack method described in the third section of this chapter, the adversary can decompose the (I_i, T) pair before encryption based on the ciphertext pair and the relevance score. First of all, due to the role of the pseudo-random function, the plaintext index and the query vector pair (P_i, Q) cannot be pushed out without knowing the key of the pseudo-random function. At the same time, even if the same two query vectors Q are used, the trapdoors will be different because of the addition of random numbers. At the same time, after random replacement, the positions of the keywords and the trapdoors will be indistinguishable, even if the decomposition gets the same trapdoor. It is also impossible to distinguish whether it is a keyword, and thus subsequent operations such as frequency statistics have no meaning.

4.2 Known Background Model

In this model, the adversary can obtain additional statistical information to infer keywords or other information. The trapdoor is represented by a $(m+U+1)$-bit vector, where the first m bits indicates whether the keyword exists in the query, and U bits will contain v random number while the other bits are 0, the last 1 bit is set to s.

First, the vector is expanded by η_r-bit random number r, which has 2^{η_r} possible values. Then use the $(m+U+1)$-bit vector S to split into two vectors. Assuming that the value of each dimension is η_q-bit and the number of 0 is μ_0, then there are $(2^{\eta_q})^{\mu_0}$ possible values. Finally, the two query vectors are encrypted with two random matrices. Therefore, the probability to distinguish two trapdoor is calculated as follows:

$$P_q = \frac{1}{2^{\eta_r} \cdot (2^{\eta_q})^{\mu_0} \cdot (2^{\eta_M})^{(n+U+1)^2 \times 2}} \tag{6}$$

It can be indistinguishable by setting a larger η_r, η_q μ_0 and η_M. For example, if $\eta_r = 1024$, $P_q < 1/2^{1024}$ and can be negligible.

4.3 Privacy

1. Data privacy. Each document will be encrypted by a symmetric encryption algorithm like AES before outsourcing. Since AES is known as semantic security [5], the adversary can not infer any information or content of the document without getting the key sk. So the confidentiality of the encrypted document can be well protected.
2. Index and trapdoor privacy. In this scheme, secure kNN algorithm is used to encrypt the index and trapdoor vector. When encrypting, both S and $\{M_1, M_2\}$ are randomly generated, so as long as the key $SK = \{S, M_1, M_2\}$ is kept secret, the cloud server cannot analyze the index or trapdoor from encrypted index and trapdoor, which has been proved secure under known ciphertext model in previous subsection.
3. Trapdoor unlinkability. In the $(m+U+1)$-bit trapdoor vector, only the keyword in "OR" operation is set to IDF value, others are random values. In addition, the trapdoor vector is scaled by random number r. Those random numbers protects the search patter, so that the trapdoor can not be distinguished even for the same query. It has been proved secure under known background model in previous subsection.

4. Keyword privacy. The $(m + U + 1)$-bit trapdoor vector consists of m-bit bloom filter, U-bit random number, and 1 bit fixed number. The bloom filter itself uses multiple locations to indicate a certain keyword, so the relationship between keywords and locations is reduced. At the same time, the random numbers ϵ_i randomize the information in the vector, so that the keyword is indistinguishable from the random number. Therefore, the statistical analysis will no longer effect.

5 Performance Analysis

To evaluate performance, we implement our scheme on real-world dataset using C# language on the Inter(R) Core(TM) i5-4590 CPU 3.30 GHz Windows 7 server.

5.1 Precision

The precision is defined as: $Precision = k^{'}/k$, where $k^{'}$ is the number of documents that actually satisfy the query, and k is the number of the documents returned. In this scenario, in addition to the relevance of the document, the false positive rate of bloom filter will also influence the precision of the results. Figure 3 gives the influence of the number of query keywords on precision. It is easy to observe that the less query keyword, the lower precision of the fuzzy query, but as the number of query keywords increases, the precision will increases since the impact of false positive rate on the results will decrease.

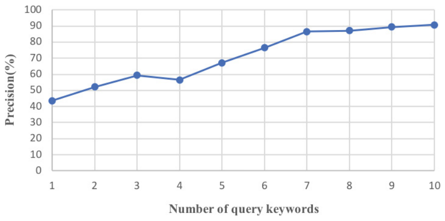


Fig. 3. Search precision

5.2 Efficiency

Index Construction. The construction of the index consists of two processes: one is to construct bloom filter for each document, and the other is to encrypt each bloom filter to generate an encrypted index. The results are shown in Fig. 4. When constructing bloom filter, the time complexity of the bloom filter construction is linearly related to the file keyword set size, the number of hash functions, and the number of documents. And in index encryption, since the encryption process uses a decomposition vector s and two secret matrices $\{M_1, M_2\}$, the complexity of the encryption depends on the size of bloom filter $\mathcal{O}(m^2)$, and since the index is generated for each document, it is also linearly related to the size of collection $\mathcal{O}(N)$. Since the process is one-time on data owner side, the time overhead is acceptable.

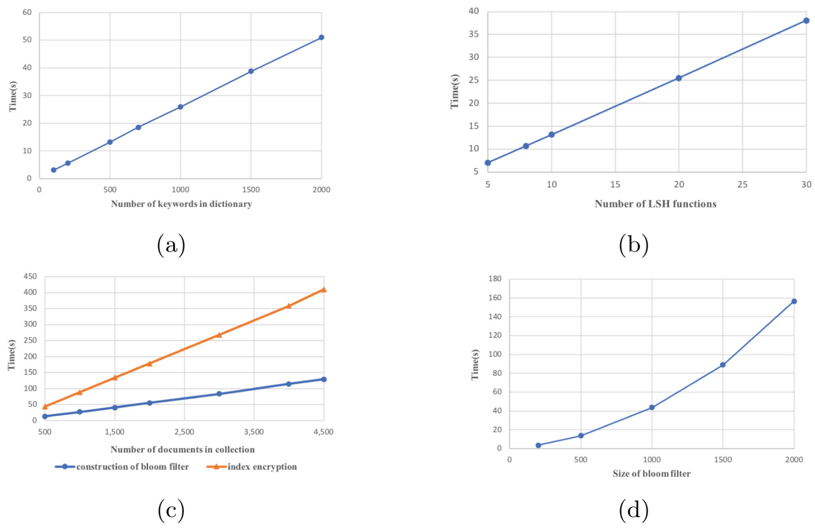
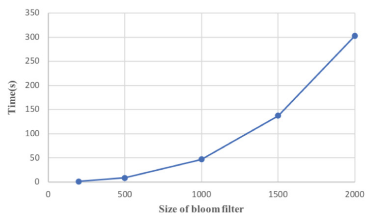


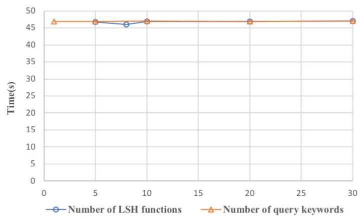
Fig. 4. The time cost of index construction. (a) The relationship between the time of bloom filter construction with the number of keywords in dictionary. (b) The relationship between the time of bloom filter construction with the number of LSH hash functions. (c) The relationship between the time of bloom filter construction and index encryption with the number of documents in collection. (d) The relationship between the time of index encryption with the size of bloom filter.

Trapdoor Generation. The complexity of the trapdoor generation depends on the secret vector S for splitting and secret matrices $\{M_1, M_2\}$. Therefore, its complexity is related to the bloom filter size $\mathcal{O}(m^2)$, as shown in Fig. 5(a), while (b) shows that the number of query keywords and hash functions has little effect on trapdoor generation. This process is generated once at the data user side and the time overhead is acceptable.

Search. The search process can be summarized as the inner product of each index vector and trapdoor vector. Therefore, the complexity depends mainly on the size of document collection and the size of the bloom filter. Figure 6(a) and (b) give the influence of the size of document collection and bloom filter, while (c) proves the number of query keywords has little influence on search.

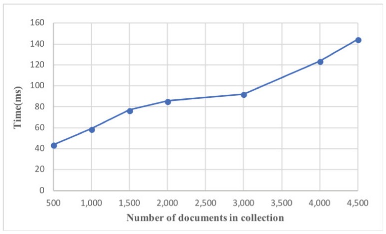


(a)

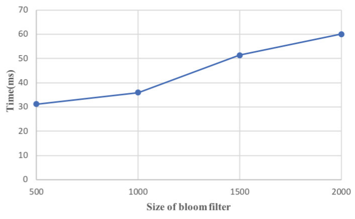


(b)

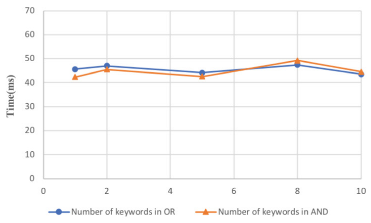
Fig. 5. The time cost of trapdoor generation. (a) For the different size of bloom filter. (b) For the different number of query keywords and LSH functions.



(a)



(b)



(c)

Fig. 6. The time cost of search. (a) For the different number of documents in collection. (b) For the different size of bloom filter. (c) For the different number of query keywords in “AND” and “OR” operations.

6 Related Work

6.1 Exact Search

Song et al. [13] first proposed a solution for searching single keyword on encrypted data with sequential scan which was provably secure but in high cost. Goh [7] defined a secure index using bloom filter and pseudo-random functions, but the scheme only support single keyword search. To provide multi-keyword function, conjunctive multi-keyword search first proposed. Boneh et al. [2] proposed a public-key scheme supporting conjunctive search and subset, range query. Wang et al. [14] designed an public-key searchable encryption scheme based on inverted index and private set intersection. Later ranked multi-keyword scheme was proposed to improve the boolean search. Cao et al. [3] first proposed a basic multi-keyword ranked search scheme (MRSE) using secure kNN computation which had low overhead on computation and communication. Chen et al. [4] used k-means algorithm to construct a hierarchical cluster index tree to develop the search efficiency.

6.2 Fuzzy Search

Fuzzy search scheme using a wildcard-based fuzzy keyword set was first proposed by Li et al. [11]. They used the edit distance to judge the keyword relevance. The scheme first constructs a wildcard-based fuzzy keyword set $S_{w_i,d}$ which contains the keyword variant with an edit distance less than d, such as $S_{CASTLE,1} = \{CASTLE, *CASTLE, *ASTLE, C * ASTLE, C * STLE, \cdots, CASTL * E, CASTL*, CASTLE*\}$ defines the variants of "CASTLE" with 1 edit distance. And then each element in the fuzzy set is encrypted. When searching, the user first generates the same encrypted fuzzy keyword set, and then the cloud server finds the corresponding fuzzy keyword and returns the corresponding file. Later, Zheng et al. [18] gave an effective attack against [11], and proved that Li et al.'s scheme [11] had low security. Wang et al. [16] made further improvements based on the work of [11] and proposed an efficient fuzzy search mechanism. The efficiency of search is improved by constructing an index tree structure based on the keywords. However, above methods must pre-configure a predefined dictionary which will cause overhead in time and storage space, and usually only support single keyword queries.

Later, locality sensitive hashing [8] was proposed to construct index which can map similar keywords to the same position to implement multi-keyword fuzzy search. At present, the research direction of fuzzy query is mainly on the innovation of the construction and functionality of fuzzy keyword index. Kuzu et al. [10] first designed the fuzzy search scheme based on Jaccard similarity, which used the bloom filter and locality sensitive hashing function minhash to map the keyword to multiple random positions to form an inverted index. The sum of the values in the same address is used to obtain the similarity between the index and the trapdoor. At the same time, the scheme also gives a multi-server public key encryption scheme, which uses the homomorphic Paillier encryption algorithm. However, due to the inverted index, the scheme only support single keyword search. Wang et al. [15] proposed the first basic scheme of multi-keyword fuzzy search based on bloom filter and locality sensitive hashing. Firstly, they transformed the keyword string into bi-gram set and used the 26^2 bits binary vector to represent the set. Later they mapped the vector into a fixed-length bloom filter by LSH function, and the relevance score is obtained by calculating the inner product of the index and the trapdoor. Fu et al. [6] improved the scheme in [15] through two main optimization. Firstly, they used stemming algorithm to extract keyword stems. Thus the keywords with same stem will be simplified to one common word which can represent more variant and provide more query request. Secondly, they selected uni-gram vector to represent the keyword. Each keyword first transformed into the uni-gram set consisting of one letter and one number to indicate the position of the letter. Later, they mapped the set into a 160-bit vector to represent the keyword. These optimizations can provide more variants of the query keyword, such as the repeated letters, and the Euclidean distance of the two keywords will be smaller which will lead to more accurate search results.

7 Conclusion

This paper proposes a multi-keyword fuzzy search scheme that supports logic queries and can resist ciphertext-only attack. We focus on the security defect of existing fuzzy search scheme, and improve the basic scheme by randomizing the bloom filter to protect the search pattern. To meet more search demands, we extend the scheme to support logic query on fuzzy search. We give a theoretical analysis of the security against two threat model and apply our scheme on real dataset to analyze the performance.

For the further work, we can extend to support semantic query and synonym query, which can scale the concept of fuzzy query, not only can consider keyword spelling errors, but also semantically similar queries. And also consider designing a scheme to support double judgments which can combine both exact and fuzzy search.

Acknowledgments. This work is supported by the National Nature Science Foundation of China (NSFC) under grant 61572026, 61672195, Open Foundation of State Key Laboratory of Cryptology (No: MMKFKT201617).

References

1. Bloom, B.H.: Space/time trade-offs in hash coding with allowable errors. Commun. ACM **13**(7), 422–426 (1970). https://ci.nii.ac.jp/naid/20001345133/en/
2. Boneh, D., Waters, B.: Conjunctive, subset, and range queries on encrypted data. In: Vadhan, S.P. (ed.) TCC 2007. LNCS, vol. 4392, pp. 535–554. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-70936-7_29
3. Cao, N., Wang, C., Li, M., Ren, K., Lou, W.: Privacy-preserving multi-keyword ranked search over encrypted cloud data. IEEE Trans. Parallel Distrib. Syst. **25**(1), 222–233 (2014)
4. Chen, C., et al.: An efficient privacy-preserving ranked keyword search method. IEEE Trans. Parallel Distrib. Syst. **27**(4), 951–963 (2016). https://doi.org/10.1109/TPDS.2015.2425407
5. Curtmola, R., Garay, J., Kamara, S., Ostrovsky, R.: Searchable symmetric encryption: improved definitions and efficient constructions. In: ACM Conference on Computer and Communications Security, pp. 79–88 (2006)
6. Fu, Z., Wu, X., Guan, C., Sun, X., Ren, K.: Towards efficient multi-keyword fuzzy search over encrypted outsourced data with accuracy improvement. IEEE Trans. Inf. Forensics Secur. **11**(12), 2706–2716 (2017)
7. Goh, E.J.: Secure indexes. Cryptology ePrint Archive, Report 2003/216 (2003). https://eprint.iacr.org/2003/216

8. Indyk, P., Motwani, R.: Approximate nearest neighbors: towards removing the curse of dimensionality. In: Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing, STOC 1998, pp. 604–613. ACM, New York (1998). https://doi.org/10.1145/276698.276876. http://doi.acm.org/10.1145/276698.276876
9. Kim, H., Park, H.: Sparse non-negative matrix factorizations via alternating non-negativity-constrained least squares for microarray data analysis. Bioinformatics **23**(12), 1495–1502 (2007)
10. Kuzu, M., Islam, M.S., Kantarcioglu, M.: Efficient similarity search over encrypted data. In: IEEE International Conference on Data Engineering, pp. 1156–1167 (2012)
11. Li, J., Wang, Q., Wang, C., Cao, N., Ren, K., Lou, W.: Fuzzy keyword search over encrypted data in cloud computing. In: Conference on Information Communications, pp. 441–445 (2010)
12. Lin, W., Wang, K., Zhang, Z., Chen, H.: Revisiting security risks of asymmetric scalar product preserving encryption and its variants. In: IEEE International Conference on Distributed Computing Systems, pp. 1116–1125 (2017)
13. Song, D., Wagner, D., Perrig, A.: Practical techniques for searches on encrypted data. In: Proceeding 2000 IEEE Symposium on Security and Privacy, SP 2000, pp. 44–55 (2000). https://doi.org/10.1109/SECPRI.2000.848445
14. Wang, B., Song, W., Lou, W., Hou, Y.T.: Inverted index based multi-keyword public-key searchable encryption with strong privacy guarantee. In: Computer Communications, pp. 2092–2100 (2015)
15. Wang, B., Yu, S., Lou, W., Hou, Y.T.: Privacy-preserving multi-keyword fuzzy search over encrypted data in the cloud. In: IEEE INFOCOM, pp. 2112–2120 (2014)
16. Wang, C., Ren, K., Yu, S., Urs, K.M.R.: Achieving usable and privacy-assured similarity search over outsourced cloud data. In: IEEE INFOCOM, pp. 451–459 (2012)
17. Wong, W.K., Cheung, D.W., Kao, B., Mamoulis, N.: Secure kNN computation on encrypted databases. In: ACM SIGMOD International Conference on Management of Data, pp. 139–152 (2009)
18. Zheng, M., Zhou, H.: An efficient attack on a fuzzy keyword search scheme over encrypted data. In: IEEE International Conference on High Performance Computing and Communications & 2013 IEEE International Conference on Embedded and Ubiquitous Computing, pp. 1647–1651 (2014)