Computers, Materials & Continua CMC, vol.65, no.2, pp.1707-1721, 2020

Paillier-Based Fuzzy Multi-Keyword Searchable Encryption Scheme with Order-Preserving

Xiehua Li[1,*], Fang Li[1], Jie Jiang[1] and Xiaoyu Mei[2]

Abstract: Efficient multi-keyword fuzzy search over encrypted data is a desirable technology for data outsourcing in cloud storage. However, the current searchable encryption solutions still have deficiencies in search efficiency, accuracy and multiple data owner support. In this paper, we propose an encrypted data searching scheme that can support multiple keywords fuzzy search with order preserving (PMS). First, a new spelling correction algorithm-(Possibility-Levenshtein based Spelling Correction) is proposed to correct user input errors, so that fuzzy keywords input can be supported. Second, Paillier encryption is introduced to calculate encrypted relevance score of multiple keywords for order preserving. Then, a queue-based query method is also applied in this scheme to break the linkability between the query keywords and search results and protect the access pattern. Our proposed scheme achieves fuzzy matching without expanding the index table or sacrificing computational efficiency. The theoretical analysis and experiment results show that our scheme is secure, accurate, error-tolerant and very efficient.

Keywords: Fuzzy multi-keywords, searchable encryption, Paillier encryption, relevance score.

1 Introduction

Data outsourcing is one of the most important applications in cloud storage and big data analysis. In many applications such as private data storage, electronic health records (EHR) systems, etc., data are very sensitive and usually involve privacy and confidential information. One naive way to protect information security is to encrypt data before outsourcing. However, data encryption reduces the flexibility and accuracy of data retrieval. The easiest solution for users to retrieve encrypted data is to download all data from the cloud storage provider (CSP) then decrypt and search locally, which is not feasible because neither the user nor the CSP could bear the massive computing and huge bandwidth usage. Aiming to this practical problem, the concept of searchable encryption is proposed. This technology is used to achieve ciphertext retrieval without revealing too much useful information.

[1] School of Computer Science and Electronic Engineering, Hunan University, Changsha, 410082, China.

[2] New Lynn School, Auckland, 0600, New Zealand.

[*] Corresponding Author: Xiehua Li. Email: beverly@hnu.edu.cn.

Received: 27 April 2020; Accepted: 29 June 2020.

The concept of searchable encryption was first proposed by Song et al. [Song, Wagner and Perrig (2000)], in which some basic search approaches over encrypted data were discussed. In the past few years, a lot of efforts have been put in designing and improving searchable encryption [Rangara, Palani and Yasminbhanu (2017); Eltayieb, Elhabob, Hassan et al. (2019); Chen, Lee, Chang et al. (2019)]. Cao et al. [Cao, Wang and Li (2013)] first defined and solved the privacy protection problem of multi-keyword ranking search on cloud data. Sun et al. [Sun, Wang and Cao (2014)] proposed a multi-keyword search scheme using a vector space model and a cosine measure with TF (word frequency)×IDF (inverse text frequency index) to provide order preserved file retrieval. Kabir et al. [Kabir and Adnan (2017)] improved Sun's scheme by writing the plaintext TF values in the index tree orderly. However, the TF values may leak the keywords and documents information. Wang et al. [Wang, Liu and Zhang (2016)] proposed a verifiable dictionary-based searchable encryption scheme, which enables users to not only search for encrypted documents, but also enable users to verify the completeness of search results. Liu et al. [Liu, Peng and Wang (2018)] proposed a verifiable diversity ranking search scheme over encrypted data that can support result verification. There are some other studies that add semantic search to searchable schemes [Xia, Chen, Sun et al. (2016)]. Also, there are many researches on searchable encryption schemes that can support multiple keywords [Yang, Li and Liu (2014); Xia, Wang and Sun et al. (2016); Peng, Lin and Yao et al. (2018); Ye, Zhou and Xu et al. (2018)].

Another issue in searchable encryption is how to solve spelling mistakes since user queries are based on their input keywords and spelling errors may cause retrieval failure. The scheme proposed by Sun et al. [Sun, Wang and Cao (2014)] can partly solve this by involving vector space model, but the search accuracy is not desirable. Other traditional spelling correction approaches like Levenshtein distance can achieve a higher accuracy only if the spelling error is less than 2 letters and is not misalignment. Some studies have also proposed fuzzy keyword search schemes. Zhou et al. [Zhou, Liu and Jing et al. (2013)] used *k*-gram to construct fuzzy keyword sets and used Jaccard coefficients to calculate keyword similarity. Gnanasekaran et al. [Gnanasekaran and Mareswari (2017)] converted keyword into a vector, and used LSH (local sensitive hash) to calculate the vector to support fuzzy keyword search. Yang et al. [Yang, Yang and Min (2017)] proposed a keyword fuzzy search scheme based on Simhash that combines Hamming distance and similarity score. Wang et al. [Wang, Li and Zhou (2017)] used sensitive hash function to index keywords, and used Bloom filter to realize fuzzy search of multiple keywords. All those schemes did not consider the misalignment of letters in the keywords, which may lead to less accurate search results.

In this paper, we proposed a Paillier-based Multi-keyword Search (PMS) scheme that supports multiple keywords fuzzy search and uses homomorphic encryption to guarantee secure keywords relevance ranking. We first proposed a Probability-Levenshtein based Spelling Correction (PLSC) algorithm, which combines word frequency and edit distance to improve the correction accuracy. Then, we used Paillier to encrypt the keywords relevance scores for each document, so that sums of keywords relevance can be calculated by the cloud server without leaking information. In addition, proxy is introduced in our scheme to support multiple DOs and multi-keyword relevance score ranking. Third, we queued the search results by the proxy to hide the connection between keywords and

downloaded files, further improving the security of the search. Our contributions can be summarized as follows.

1) To the best of our knowledge, we first proposed a spelling correction algorithm that uses both probability and Levenshtein algorithm to improve the correction accuracy. We also experimented on the PLSC accuracy in a chosen document library.

2) To support encrypted multi-keywords order-preserving search we used Paillier to encrypt the keywords relevance scores and outsource relevance score calculation to the cloud server.

3) We built up a system that used proxy to support multiple DOs and used queuing to break the connection between user queries and associated documents. We implemented our scheme and the results demonstrated the accuracy and efficiency of our scheme.

The rest of the paper is organized as follows. Section 2 presents the constructions and definitions of our scheme. Section 3 describes the definition of basic functions used for supporting multi-keywords fuzzy search. Theoretical system performance is analyzed in Section 4. We give the scheme implementation results and comparison in Section 5. Section 6 is the conclusion of the whole paper.

2 Constructions and definitions

2.1 System model

Fig. 1 shows the system structure of this scheme. Before outsourcing data to cloud server, the data owner (DO) builds indexes and uses AES to encrypt files. Then DO uploads encrypted files and their associated indexes to the Proxy. Proxy builds and uploads the secured indexes with the encrypted files to the Cloud Server. Meanwhile, DO distributes file decryption keys to authorized users via secure channel.

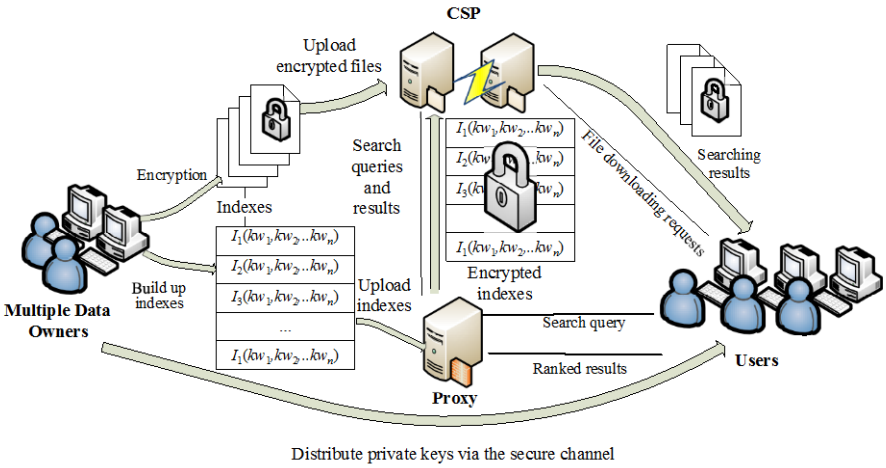


Figure 1: System structure of encrypted files fuzzy searching

2.2 Security assumptions

We adopt the "honest-but-curious" model for the cloud server as most of the schemes did. It assumes that the cloud server can honestly implement the program but is willing to get

illegal profits if given the opportunity. Also, the cloud server may analyze the stored data to learn more information. In addition, we assume that the DO is honest and has no interest in collusion attack with cloud service providers. This assumption is reasonable because DO has the original plaintext and index information. There is no need for the DO to collude with others to get information. Users here are not trusted entities; they may collude with cloud server or other users to get more information about the encrypted data. In this scheme, Proxy is a trusted entity who receives plaintext search queries from users, then encrypts and forwards those queries to Cloud Server; it receives the results from Cloud Server, decrypts relevance scores and ranks the results before sending users the search results.

2.3 Scheme goals

In order to achieve multi-keyword fuzzy search on encrypted data, 3 important goals need to be fulfilled:

1) **Security:** keep the indexes secure and break the linkability of indexes and files, so that the cloud server and other malicious entities will not get valuable information about the encrypted data;

2) **Fuzzy searchable:** automatic and accurate spelling correction should be supported since misspelling happens all the time;

3) **Multi-keyword ranked search:** the new scheme should support multi-keyword ranked search and be both time and space efficient.

2.4 Notations

The notations used in this paper are defined as followed:

- D: plaintext documents set, $D=\{D_1, D_2, ..., D_n\}$;
- D': encrypted documents set, $D'=\{D_1^{'}, D_2^{'}, ..., D_n^{'}\}$;
- ID: document identifier in plaintext $ID=\{id_1, id_2, ..., id_n\}$;
- ID': document identifier in ciphertext, $ID^{'}=\{id'_1, id'_2, \ldots, id'_n\}$;
- W: keywords set in plaintext, $W=\{W_1, W, \ldots, W_m\}$;
- W': keywords set in ciphertext, $W'=\{W_1^{'}, W_2^{'}, ..., W_n^{'}\}$;
- $S_{i,j}$: plaintext relevance score of keyword W_i in document D_j; S_j: sum of relevance score in document j in plaintext;
- $S_{i,j}^{'}$: encrypted relevance score of keyword W_i in document D_j; $S^{'}{}_j$: sum of relevance score in document j in ciphertext;
- **Keygen1** (): AES key generation algorithm;
- **Keygen2** (): Paillier key generation algorithm, (PK, SK) are the public and secret key pairs generated by **Keygen2 ()**.
- **Enc1** (m, k): message m is encrypted by symmetric key k with AES encryption

algorithm; **Dec1** (c, k) means ciphtertext c is decrypted by key k with AES algorithm.

- **Enc2** (m, pk): message m is encrypted by public key pk with Paillier encryption algorithm; **Dec2** (m, sk) means ciphtertext c is decrypted by key k with AES algorithm.

3 Spelling correction algorithm

Misspelling happens in many ways: wrong letters, missing letters, redundant letters, jumbled letters. We proposed the Probability-Levenshtein based spelling correction (PLSC) to correct misspelling in a more precise way.

3.1 Probability-Levenshtein based spelling correction (*PLSC*)

The PLSC algorithm is executed during the keywords input phase. According to user's input, several correction suggestions of keywords will be given to user. In the initiation process, a statistical vocabulary rule set is built based on a chosen article library. Usually, if the word is misspelled, at least some of the letters should be spelled correctly. Therefore, our scheme is constructed based on the correct input letters. Considering the input habits, we suppose that for each input word of length N, at least N-2 characters are correct input, and the first letter is correct. We calculate the frequency of each word in the chosen article library. Moreover, we add up the frequencies of each combination with the length [2, N]. For example, for the word *beat*, its frequency in the chosen library is f (*beat*), all possible combinations of *beat* are (*be*, *ba*, *bt*, *bea*, *bet*, *bat*, *beat*). For the combination *bea*, it also appears in other words like *bear*, *bean*, etc., thus the frequency of *bea* is defined as f (*bea*)=f (*beat*)+f (*bean*)+…. In order to give more precise candidate keywords, we define a Derivative Possibility (DP) to describe the probability of deriving a word from a certain letter combination in the article library.

Definition: Let P (W/w_i) be the Derivative Possibility of W from w_i, $w_i \in W$ is a possible letter combination derived from W.

$$P_i = \frac{freq(W_t)}{freq(w_i)} \tag{1}$$

where *freq* (W_t) is the frequency of a word W_t in the chosen article library, *freq* (w_i) is the summation frequency of a combination w_i. Rule set is established in the system initialization stage, the frequency of various letter combinations in the article library and their P_i are saved in the rule set. In order to improve query efficiency, we organize the rule set with hash table. When user inputs a keyword, PLSC compares the input with the items in the rule set and gives the user the top-k candidates with the highest P_i.

However, based on our observation the spelling correction algorithm cannot solve the misspelling problem only with DP. Consider the following situation. When the word *stand* is misspelled as *stanf*, and if *freq* (*staff*)/*freq* (*stanf*) is higher than *freq* (*stand*)/*freq* (*stanf*), then the recommended priority of *stand* will be lower than *staff*. Obviously, this result is not accurate enough. In addition, when the same letter combination introduces different words with similar or even equal probabilities, it will be difficult to determine the result of the error correction. Based on our observation, we add input similarity

comparison on the statistical word error correction algorithm to provide more accurate recommendation words. We adopt the edit distance algorithm to calculate the similarity of two stings A and B with length N_1 and N_2 respectively.

$$Sim(A,B) = 1 - \frac{ED(A,B)}{\max(N_1, N_2)} \tag{2}$$

When a user inputs a word, the input combination set C_p is extracted, and each item in C_p is compared with the rules in the rule set to obtain a possible word set $W=\{W_1, W_2, ..., W_n\}$, and the corresponding probability set $\{P_1, P_2, ..., P_n\}$. Then, the similarity of input word and words in W will be compared to get the final recommendation results. We user *Score* (W_i) to identify the accuracy of each recommended word.

$$Score(W_i) = \alpha P_i + \beta Sim(W_i, input), \alpha + \beta = 1 \tag{3}$$

where α, β are constants, and $\alpha+\beta=1$. According to *Score* (W_i), top-k recommended results will be returned.

3.2 Error correction schemes comparison

We choose 1600 articles published on USENIX in the last 5 years as the article library, and calculate the frequency of each word in the library. Rule set is built based on each word, its combination, word frequency and P_i. The value of α, β can be defined by users. We implemented experiment to test the correction accuracy with different α and β values. In the experiment, we select 1000 words with the highest frequency and put 1 to 2 random error in each word, the error includes error letter, missing or extra letters. The selected words are composed of more than 5 letters. For each word the system will give top-5 correction candidates with the highest *Score* (W_i). If the i^{th} candidate is the correct word, the value of this correction is 5-i. We test the correction accuracy of Eq. (3) with different values of α and β, the test results show that words error correction algorithm is more accurate when α=0.3, β=0.7. Tab. 1 shows the correction values comparison among our PLSC algorithm, edit distance and Norvig's spelling corrector with difference number of random errors.

Table 1: Score comparison with random spelling errors

Errors	PLSC	Norvig's	Edit distance
1～2	4739	4473	4257
2	4660	4057	3672
1～3	3578	3240	3005

From Tab. 1 we can see that our PLSC algorithm has more accuracy than Norvig's algorithm and edit distance especially when there are more than 2 random errors in the word.

4 Multi-keyword order preserving searchable encryption

Our multi-keyword order-preserving searchable encryption scheme includes three major processes: index building, ciphertext searching and queue-based ciphertext retrieval.

4.1 Index building

Index building process is run on both DO and Proxy. DO is responsible for building index for each file and encrypting files. Proxy is responsible for collecting indexes and associated encrypted files from all DOs. Proxy needs to combine indexes and files, and then build up secured indexes for all files.

DO Process. DO extracts keywords from the documents and calculates the relevance scores of all keywords in each file with TF-IDF algorithm. We use keyword indexing to arrange documents. DO builds up the plaintext index I for all documents, where $I=\{I_1, I_2, I_3, ..., I_m\}$ and $I_i=(W_i,<id_{j(1\leqslant j\leqslant n)}, S_{i,j(1\leqslant j\leqslant n)}>)$. I_i is the index of keyword W_i, and $id_{j(1\leqslant j\leqslant n)}$ is the document which contains W_i. $S_{i,j(1\leqslant j\leqslant n)}$ denotes the relevance score of keyword W_i in document j.

DO runs **Keygen1 ()** to get the document encryption key K_1, and then runs **Enc1 ()** to encrypt documents and their identifiers with K_1.

$$C = \{(id'_1, D'_1), (id'_2, D'_2), \ldots, (id'_n, D'_n)\} \tag{4}$$

where D'_i=**Enc1** (D_i, K_1), id'_i=**Enc1** (id_i, K_1), $i \in \{1,2,...,n\}$; DO then uploads (I, C) pair to proxy for further processing.

DO finally distributes decryption keys to authorized users.

Proxy Process. To support multi-DO application scenario, this scheme uses Proxy to complete index encryption. After accepting the (I, C) pairs from DOs, Proxy generates the encrypted index I' and sends (I', C) pair to the CSP.

Proxy runs **Keygen1 ()** and **Keygen2 ()** to get the AES key and Paillier key pair respectively.

$$\begin{aligned}
&K_2 \leftarrow \textbf{Keygen2()},\ K_2=(PK,SK);\ K_3 \leftarrow \textbf{Keygen1()} \\
&W'_i = \textbf{Enc1}(W_i, K_3) \\
&id''_{j(1\leq j\leq n)} = \textbf{Enc1}(id_{j(1\leq j\leq n)}, K_3) \\
&S'_{i,k} = \textbf{Enc2}(S_{i,k}, PK), \\
&I' = \{I'_1, I'_2, ..., I'_i\}, I'_i = (W'_i, id''_j, S'_{i,k})
\end{aligned} \tag{5}$$

4.2 Ciphertext searching

User starts searching process by putting in a set of search keywords. The client executes PLSC algorithm to correct input errors and sends the keywords set $SW=\{W_1, W_2, ..., W_t\}$ to the Proxy. The Proxy then uses K_3 to generate the trapdoor and submits $SW'=\{W'_1, W'_2, ..., W'_t\}$ to CSP for searching. In this section we use an example to describe the complete process of our order preserving ciphertext searching algorithm.

CSP Searching Algorithm

CSP performs a ciphertext searching algorithm based on SW' and I'. After finding documents that contain all keywords in SW', CSP adds up the total relevance score of all keywords in each document. The detailed searching algorithm is described in Fig. 2.

For example, user puts in a keyword set $SW=\{W_a, W_b\}$ and sends SW to the Proxy. Proxy generates the trapdoor $SW' =\{W'_a, W'_b\}$ and sends it to CSP. CSP then searches the entire I'. Suppose CSP gets the result:

$$I'_a = \{(W'_a, id''_1, S'_{a,1}), (W'_a, id''_2, S'_{a,2}), (W'_a, id''_3, S'_{a,3})\}$$

$$I'_b = \{(W'_b, id''_2, S'_{b,2}), (W'_b, id''_3, S'_{b,3}), (W'_b, id''_4, S'_{b,4})\}$$

$$I'_m = \{I'_a \cap I'_b\}$$

CSP then chooses the documents that contain both W'_a and W'_b and puts them in set $I'_r = \{W'_a, id''_2, S'_{a,2}, W'_a, id''_3, S'_{a,3}, W'_b, id''_2, S'_{b,2}, W'_b, id''_3, S'_{b,3}\}$. CSP sums up the relevance scores for W'_a and W'_b in id''_2 and id''_3 with Paillier algorithm. The final relevance score $S'_2 = S'_{a,2} * S'_{b,2}$ and $S'_3 = S'_{a,3} * S'_{b,3}$. Finally, CSP returns the searching results $\{(id''_2, S'_2), (id''_3, S'_3)\}$ to Proxy.

Order Preserving Algorithm

By receiving $\{(id''_2, S'_2), (id''_3, S'_3)\}$ from CSP, Proxy runs **Dec1** () to decrypt the document identifier and **Dec2** () to decrypt the relevance score.

$id_2 \leftarrow$ **Dec1** (id''_2, K_3), $id_3 \leftarrow$ **Dec1** (id''_3, K_3)

$S_2 \leftarrow$ **Dec2** (S'_2, SK), $S_3 \leftarrow$ **Dec2** (S'_3, SK)

Proxy ranks the documents by the relevance score S_i and returns the top-k document identifiers to user. Before downloading documents from the CSP, user needs to encrypt documents identifiers with K_1 under Enc1 () and gets id'_2=Enc1 (id_2, K_1), id'_3=Enc1 (id_3, K_1). User sends (id'_2, id'_3) to the CSP and downloads documents (D'_2, D'_3). Finally, users decrypt documents with K_1 under Dec1 () and get D_2=Dec1 (D'_2, K_1), D_3=Dec1 (D'_3, K_1).

Algorithm Ciphertext search and rank

For the input keywords $SW' = \{W'_1, W'_2, ..., W'_t\}$
For $(x \leqq t)$
{Search I';
  If $(W'_x \in I')$
  {$I'_m = I'_m \cup I'_x$;
   While $(I'_m \neq \varnothing)$
    {Puts the documents in I'_m that contains all keywords in SW' to I'_r;
     Calculate the total relevance score of all keywords in SW' for each document;
     Send I'_r and summarized relevance score to Proxy;
    }
  }
}

Figure 2: CSP searching algorithm

4.3 Queuing-based ciphertext retrieval

Current state-of-art searchable encryption schemes have two main drawbacks: lack of protection on file-access pattern (in which files are returned in response to each query) and leakage of query pattern (when a query is repeated [Zhang, Kazt and Papamanthou (2016)]. In our scheme, we avoid the first drawback by breaking the linkability between keywords and files. For the second drawback, we use the queuing-based ciphtertext retrieval to mess up the query pattern.

As an honest but curious principal, CSP tries to obtain information about the linkability of keywords, indexes and documents by analyzing each query. To break down the linkability, we encrypt the file identifiers in the index and ciphertext documents with different keys. However, CSP can still obtain the connection of keywords and associated files by observing which files are downloaded after a certain query. For example, the proxy forwards a query for the keyword W'_i, and then a user downloads some files from CSP. With few rounds of such query and downloading, CSP could get the connection between W'_i and ciphtertext files that contain W'_i. To hide the relationships between user queries and downloaded files, we propose a queuing-based ciphertext retrieval scheme that uses Proxy to queue queries. In this scheme, after receiving research results from CSP, Proxy will wait till a random number of results have arrived and then forwards all results to users simultaneously. In this case, multiple users will request for documents downloading at the same time, and the connection between keywords and file identifiers can be hidden. In order to keep the balance between system efficiency and search security, the queuing strategy will be divided into two parts: ① By setting the timestamp T_{max}, the maximum waiting time of the Proxy after receiving a retrieval result is determined. ② Set the minimum number of queued queries K_{min}, Proxy waits for at least K_{min} search results return. If the Proxy queuing time $t<T_{max}$ and the queued requests $K > K_{min}$, the Proxy will forward all queries. Whereas, if $t>T_{max}$ and $K<K_{min}$ the Proxy will forge some fake requests and forward them along with the normal requests.

5 Security analysis

5.1 Data confidentiality

In our scheme, the plaintext documents are encrypted before outsourcing to the cloud and the decryption keys are distributed to users by DO via a secure channel. Hence, documents security can be achieved. Index is built by DO and encrypted by the Proxy. In addition, we encrypt the index and file identifier with different keys (K_3 and K_1) so that the CSP cannot obtain the connection between the encrypted documents and the encrypted index. Keywords relevance scores for each document are encrypted with Paillier encryption. Even though the encrypted relevance scores are accumulated by the CSP, the CSP cannot get any information about keywords or their relevance scores. Therefore, as long as the Proxy and users keep their respective keys properly, the confidentiality of data, index, keyword information and relevance scores can be guaranteed.

5.2 Search security

When users send out search queries, they send the keywords to the Proxy, then the Proxy encrypts and forwards the queries to the CSP. Since the queries are encrypted and forwarded by the Proxy, the CSP can get neither the keywords nor user information, hence user privacy can be protected. Meanwhile, in order to hide the connection between keywords and documents, all query results are queued by the Proxy before being forwarded to users. Suppose in time slot T a number of query results are received by the Proxy from CSP is N. These queries belong to M users. Finally, CSP will receive M groups of documents downloading requests from users. Therefore, N groups of encrypted keywords searching can be matched with M groups of documents. i) If $N=M$, every user only sends one search request. After queuing on the Proxy, the possibility of CSP successfully guessing the correspondence between keywords and documents will be as followed.

$$\frac{1}{A_M^M} = \frac{1}{M!} \tag{6}$$

ii) If $N>M$, which means at least one user sends multiple search queries or contains forged search requests: ① If N requests are from users, according to the second type of Stirling number, the number of schemes for dividing the N sets of search requests into M groups will be:

$$S(N,M) = \frac{1}{M!}\sum_{k=0}^{M}(-1)^k\binom{M}{k}(M-k)^N \tag{7}$$

Then the probability of CSP successful guessing will be:

$$P = \frac{1}{S(N,M) * A_M^M} = \frac{1}{M! * S(N,M)} \tag{8}$$

② If k ($0<k\leq N-M$) search requests are forged by the Proxy, the CSP successful guessing rate will be:

$$P = \frac{1}{S(N,M) * A_M^M} * M^k = \frac{M^k}{M! * S(N,M)} \tag{9}$$

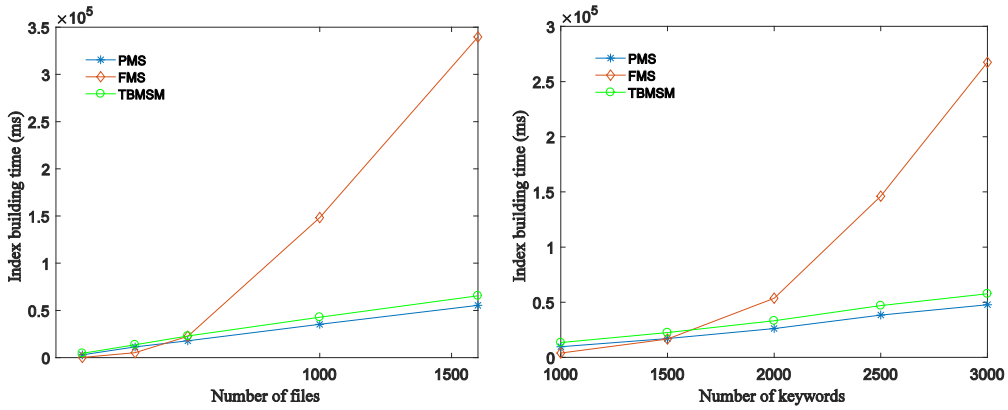
From Eqs. (7) to (9), we can deduce that by queuing and forging user search queries. The possibility of CSP successfully guessing the connection between search keywords and their associated documents would be very low, especially when M and N are large. It is safe to conclude that with the increase of N, the probability P decreases significantly and continuously. If a user sends multiple search requests in a short period of time, the queuing query through the Proxy can greatly improve the guessing difficulty of the CSP. If the Proxy sends a forged search request, and since the search request contains a forged search request, the CSP cannot guess the correspondence between the completely correct keyword and the file. Therefore, setting the proxy queuing query and forging the search request can effectively prevent the CSP statistical guessing attack and improve security.

6 Performance evaluation

Our performance experiment is implemented on Windows 7 with Intel (R) Core (TM) i5 6500 3.2 GHz and 2GB RAM. Our implementation is in C++ language. We select 1600 papers published in USENIX in the last 5 years and extract 3000 keywords. We evaluate the scheme performance in the following ways: (1) index generation efficiency; (2) trapdoor generation efficiency; (3) retrieval efficiency. We compare our scheme with the most relevant researches on searchable encryption-FMS [Li, Yang and Luan (2016)] and TBMSM [Peng, Lin, Yao et al. (2018)] schemes.

6.1 Index building efficiency

In this section, we compare the index building time and storage cost among our scheme, the FMS scheme and the TBMSM scheme. Fig. 3(a) shows the time overhead required to build an index with the number of documents ranging from 100 to 1600. It can be seen that as the number of files grows, the index generation time of our PMS scheme increases slowly. When the number of files is smaller than 500, the efficiencies of PMS, FMS and TBMSM are almost the same, with FMS being better than PMS and TBMSM. But the difference is not significant. However, if the number of files is bigger than 500, the index building time of FMS grows dramatically. Compared with TBMSM, PMS has a slightly less index building time. Fig. 3(b) shows the time required for generating document index with the number of keywords ranging from 1000 to 3000. We can see that FMS has a greater growth rate with the increase number of keywords. While the time cost on building index with PMS scheme is stable and slowly increases, the growth rate stays basically unchanged. The index generation efficiency of the PMS exceeds FMS at approximately 1500 keywords. With the growth of document numbers and keywords, the index building efficiency of PMS becomes better than that of TBMSM.



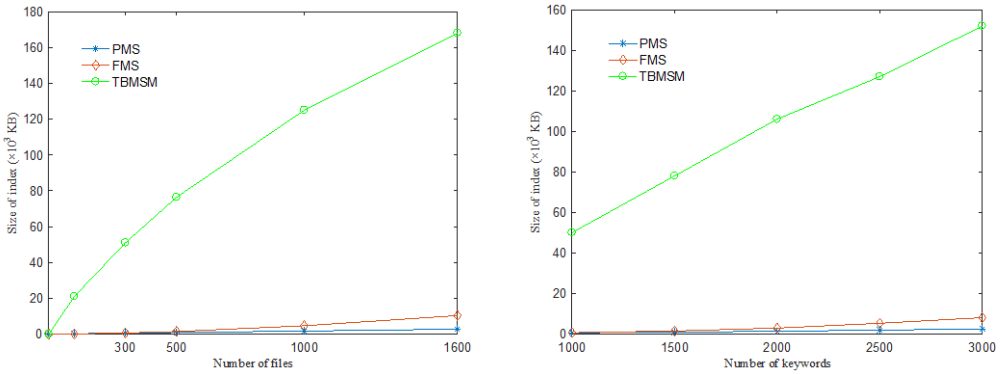
a) Number of documents b) Number of keywords

Figure 3: Index building time

Fig. 4(a) shows the space required for storing indexes with the number of files ranging from 100 to 1600. Fig. 4(b) shows the space required for storing indexes with the number of keywords ranging from 1000 to 3000. When the number of files is greater than 300 or the number of keywords in the index is greater than 1000, the index storage overhead of

the PMS scheme is better than that of the FMS scheme, and the growth rate of PMS is slower than that of FMS. The index storage overhead of TBMSM is much larger than that of PMS. When the number of keywords in the index reaches 2000, the index storage space of TBMSM exceeds 100 MB.

In real applications, especially in cloud storage systems, there would be a massive number of stored documents and keywords. Our PMS scheme is more efficient both in index generation and storage than the other two schemes.



a) Number of documents b) Number of keywords

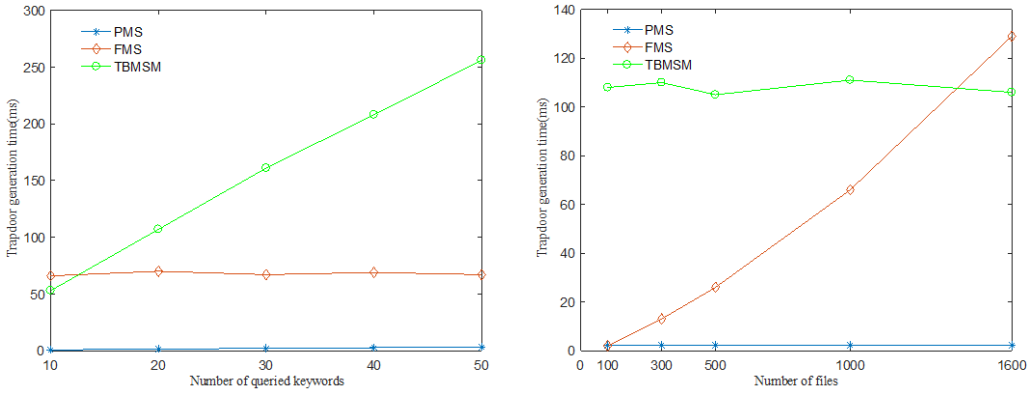
Figure 4: Index storage space

6.2 Trapdoor generation efficiency

This section compares the trapdoor generation efficiency of the three schemes discussed before.

Fig. 5(a) shows the trapdoor generation efficiency over 1000 files with queried keywords ranging from 10 to 50. Fig. 5(b) shows the trapdoor generation efficiency on 20 keywords with the number of files ranging from 100 to 1600.

In Fig. 5(a) we can see that the trapdoor generation time in FMS is not affected by the number of search keywords. The trapdoor generation time in our PMS and the TBMSM scheme is only related to the number of queried keywords. With the growth of keywords, the trapdoor generation time in TBMSM grows linearly while PMS grows slowly regarding to the number of search keywords. In Fig. 5(b), the number of query keywords is set to be 20. The trapdoor generation time of FMS grows linearly with the increase of files number, while the TBMSM and our PMS schemes remain stable. The reason why FMS has a linear growth is that the trapdoor is generated with matrix operation ($M_1^{-1}qa$, $M_2^{-1}qb$). Therefore, as the number of files and keywords in the index increases, the dimension of the encryption matrix increases accordingly, which causes the linear growth of trapdoor generation time. The reason why TBMSM uses more time on trapdoor generation is that they use the bilinear mapping for different users. From the two figures, it can be clearly seen that the PMS scheme has better trapdoor generation efficiency than the FMS and TBMSM schemes, and the PMS efficiency is more significant than the FMS efficiency as the number of files increases.



a) Number of keywords in queries b) Number of files with 20 keywords

Figure 5: Trapdoor generation efficiency

6.3 Search efficiency

This section compares the search time overhead of PMS, FMS and TBMSM schemes. All schemes are tested with the number of files ranging from 100 to 1600, and the number of queried keywords is set to be 5 per query.

As shown in Fig. 6, our PMS scheme has less search time than the FMS and TBMSM schemes. The search time of PMS is less than 1s even though there are 1600 encrypted papers in the database. The search time in the FMS and TBMSM schemes increases as the number of files increases.

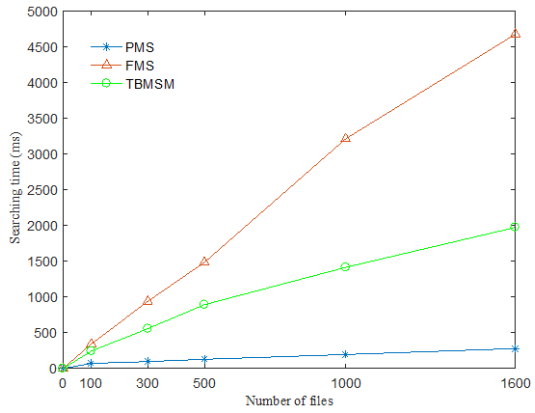


Figure 6: Search efficiency

7 Conclusion

Our scheme presents a searchable encryption scheme that can support fuzzy multiple keywords order-preserved searches. In this paper, a Probability-Levenshtein spelling correction algorithm is proposed to correct user input errors and fuzzy keywords searches. This algorithm combines both the keyword appearance frequency and Levenshtein

distance to improve the error correction accuracy, and further to improve the success rate and accuracy of document retrieval. Then we adopt the Paillier homomorphic encryption to encrypt keywords relevance score, which can support order preserving searches and outsource the expensive computational cost to CSP. Furthermore, we introduce the Proxy in our scheme to handle the encryption of indexes for multiple DOs and securely forward user search queries. Also, the Proxy uses a queue-based algorithm to break the connection between user queries and documents downloading. Based on our theoretical security proof, our scheme can guarantee both data security and search security. Finally, we implement and compare our scheme with another two close relevant schemes, where the implementation results show that our PMS scheme is more efficient in many ways.

Acknowledgement: This work is supported by the National Natural Science Foundation of China under Grant 61402160 and 61872134. Hunan Provincial Natural Science Foundation under Grant 2016JJ3043. Open Funding for Universities in Hunan Province under grant 14K023.

Funding Statement: 1. National Natural Science Foundation of China under grant 61402160 with Xiehua Li as PI, and 61872134 with Xiehua Li as Co-PI. URL to the sponsor's website is: http://www.nsfc.gov.cn/; 2. Hunan Provincial Natural Science Foundation under grant 2016JJ3043 with Xiehua Li as PI. URL to the sponsor's website is: http://kjt.hunan.gov.cn/; 3. Open Funding for Universities in Hunan Province under grant 14K023 with Xiehua Li as PI. URL to this website is: www.hnedu.cn.

Conflicts of Interest: There is no conflicts of interest to report regarding the present study.

References

Cao, N.; Wang, C.; Li, M. (2013): Privacy-preserving multi-keyword ranked search over encrypted cloud data. *IEEE Transactions on Parallel & Distributed Systems*, vol. 25, no. 1, pp. 222-233.

Chen, L.; Lee, W. K.; Chang, C.; Choo, K. R.; Zhang, N. (2019): Blockchain based searchable encryption for electronic health record Sharing. *Future Generation Computer Systems*, vol. 95, no. 1, pp. 420-429.

Eltayieb, N.; Elhabob, R.; Hassan, A.; Li, F. (2019): An efficient attribute-based online/offline searchable encryption and its application in cloud-based reliable smart grid. *Journal of Systems Architecture*, vol. 98, no. 1, pp. 165-172.

Gnanasekaran, P.; Mareswari, C. (2017): A secure and dynamic multi-keyword ranked search scheme over encrypted cloud data. *International Research Journal of Advanced Engineering and Science*, vol. 2, no. 3, pp. 70-75.

Kabir, T.; Adnan, M. A. (2017): A dynamic searchable encryption scheme for secure cloud CSP operation reserving multi-keyword ranked search. *Proceedings of the 4th International Conference on Networking, Systems and Security*, Dhaka, Bangladesh, pp. 1-9.

Li, H.; Yang, Y.; Luan, T. H. (2016): Enabling fine-grained multi-keyword search supporting classified sub-dictionaries over encrypted cloud data. *IEEE Transactions on Dependable and Secure Computing*, vol. 13, no. 3, pp. 312-325.

Liu, Y.; Peng, H.; Wang, J. (2018): Verifiable diversity ranking search over encrypted outsourced data. *Computers, Materials & Continua*, vol. 55, no. 1, pp. 37-57.

Peng, T.; Lin, Y.; Yao, X.; Zhang, W. (2018): An efficient ranked multi-keyword search for multiple data owners over encrypted cloud data. *IEEE Access*, vol. 6, no. 1, pp. 21924-21933.

Rangaraj, A. M.; Palani, S.; Yasminbhanu, P. (2017): A secure and dynamic multi-watchphrase ranked search scheme over encrypted cloud data. *International Journal of Innovative Technology and Research*, vol. 5, no. 2, pp. 5933-5947.

Song, D. X.; Wagner, D.; Perrig, A. (2000): Practical techniques for searches on encrypted data. *Proceeding of IEEE Symposium on Security and Privacy*, Berkeley, CA, USA, pp. 44-45.

Sun, W.; Wang, B.; Cao, N. (2014): Verifiable privacy-preserving multi-keyword text search in the cloud supporting similarity-based ranking. *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 11, pp. 2025-3035.

Wang, K.; Li, Y.; Zhou, F. (2017): Fuzzy ciphertext search scheme for multi-keywords. *Journal of Computer Research and Development*, vol. 54, no. 2, pp. 348-360.

Wang, S.; Liu, L.; Zhang, Y. (2016): Verifiable dictionary based searchable encryption scheme. *Journal of Software*, vol. 27, no. 5, pp. 1301-1308.

Xia, Z.; Chen, L.; Sun, X.; Liu, J. (2016): A multi-keyword ranked search over encrypted cloud data supporting semantic extension. *International Journal of Multimedia and Ubiquitous Engineering*, vol. 11, no. 8, pp. 107-120.

Xia, Z.; Wang, X.; Sun, X.; Wang, Q. (2016): A secure and dynamic multi-keyword ranked search scheme over encrypted cloud data. *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 2, pp. 340-352.

Yang, Y.; Li, H.; Liu, W.; Yang, H.; Wen, M. (2014): Secure dynamic searchable symmetric encryption with constant document update cost. *Proceedings of IEEE Global Communications Conference*, Austin, TX, USA, pp. 775-780.

Yang, Y.; Yang, S.; Min, K. (2017): Simhash-based fuzzy ranked search scheme over encrypted cloud data. *Chinese Journal of Computers*, vol. 40, no. 2, pp. 431-444.

Zhang, Y.; Katz, J.; Papamanthou, C. (2016): All your queries are belong to us: the power of file-injection attacks on searchable encryption. *Proceedings of the 25th USENIX Security Symposium*, Austin, TX, USA. pp. 707-720.

Zhou, W.; Liu, L.; Jing, H.; Zhang, C.; Yao, S. et al. (2013): K-gram based fuzzy keyword search over encrypted cloud computing. *Journal of Software Engineering and Applications*, vol. 6, no. 1, pp. 29-32.