Received February 2, 2019, accepted March 5, 2019, date of publication March 13, 2019, date of current version April 2, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2904763

Dynamic Multi-Keyword Ranked Search Based on Bloom Filter Over Encrypted Cloud Data

CHENG GUO[1,2], RUHAN ZHUANG[1,2], CHIN-CHEN CHANG[3], AND QIONGQIONG YUAN[1,2]

[1]School of Software Technology, Dalian University of Technology, Dalian 116620, China
[2]Key Laboratory for Ubiquitous Network and Service Software of Liaoning Province, Dalian 116620, China
[3]Department of Information Engineering and Computer Science, Feng Chia University, Taichung 40724, Taiwan

Corresponding author: Chin-Chen Chang (alan3c@gmail.com)

This work was supported by the National Natural Science Foundation of China under Grant 61871064, Grant 61501080, Grant 61771090, and Grant 61601214.

ABSTRACT Cloud computing has become a popular approach to manage personal data for the economic savings and management flexibility in recent year. However, the sensitive data must be encrypted before outsourcing to cloud servers for the consideration of privacy, which makes some traditional data utilization functions, such as the plaintext keyword search, impossible. To solve this problem, we present a multi-keyword ranked search scheme over encrypted cloud data supporting dynamic operations efficiently. Our scheme utilizes the vector space model combined with TF $\times$ IDF rule and cosine similarity measure to achieve a multi-keyword ranked search. However, traditional solutions have to suffer high computational costs. In order to achieve the sub-linear search time, our scheme introduces Bloom filter to build a search index tree. What is more, our scheme can support dynamic operation properly and effectively on the account of the property of the Bloom filter, which means that the updating cost of our scheme is lower than other schemes. We present our basic scheme first, which is secure under the known ciphertext model. Then, the enhanced scheme is presented later to guarantee security even under the known background model. The experiments on the real-world data set show that the performances of our proposed schemes are satisfactory.

INDEX TERMS Cloud computing, dynamic searchable encryption, multi-keyword ranked search, Bloom filter.

I. INTRODUCTION

With the development of cloud computing, more and more people realize the benefits that can be reaped from it. Meanwhile, people have to manage a massive data in this era of information explosion, which may not only increase the management cost but also lose efficiency. To solve this problem, people, companies or organizations can take advantage of cloud computing, which can enable convenient and on-demand network access to a shared pool of configurable computing resources [1]. More specifically, data owners can outsource their data into cloud servers so that they can get access to these data as they want. Obviously, cloud computing takes the economic savings and management flexibility for us. However, most data owners are not willing to store their data, especially for some sensitive data such as financial records and personal emails into cloud servers on account of the data privacy. The cloud server is semi-trusted, which may leak data privacy. So it is necessary for data owners to encrypt their outsourced data. Unfortunately, encryption can reduce the efficiency of data utilization, especially for some services that are based on plaintext keyword search.

For the above issues, searchable encryption can provide some useful techniques for cloud services on the basic of keyword search. Searchable encryption allows users to get access to relevant data by searching their encrypted data. The first searchable encryption scheme was proposed by Song *et al.* [2]. Their scheme utilizes two-layered encryption, which can guarantee the correctness of the trapdoor. Although this scheme is proven to be secure, it is based on a weak security model. To address the drawbacks of Song *et al.* [2], some novel searchable encryption schemes [3]–[10] were proposed. These schemes build encrypted indices for searching rather than searching on the encrypted data directly. Curtmola *et al.* [4] introduced two new adversarial models for searchable encryption, called chosen-keyword attacks (CKA1) and adaptive chosen-keyword attacks (CKA2). They are widely used as the standard definitions to date. And two

The associate editor coordinating the review of this manuscript and approving it for publication was Abid Khan.

secure schemes are proposed against CKA1 and CKA2, respectively. Kamara *et al.* [10] presented a dynamic searchable symmetric encryption scheme. Their scheme can satisfy their proposed requirements of the more strict security definition and support dynamic operation, which makes the search more flexible.

However, the above searchable encryption schemes can only support exact single keyword search, which restricts the application range of the solutions. The data owner prefers to multi-keyword search rather than single keyword search, and the former can improve the search's accuracy. So the multi-keyword search has been researched extensively recently. The existing multi-keyword search schemes [11]–[14] can realize many multi-keyword search related functions such as conjunctive keyword search, disjunctive keyword search and subset search. Ballard *et al.* [11] proposed two different conjunctive keyword search schemes, which only return the files containing all the searched keywords, on the basis of Shamir secret sharing and bilinear pairings, respectively. Their scheme is proven secure in the standard model. And disjunctive keyword search scheme was proposed in [14] later, which can return files containing the subset of query keywords. Meanwhile predicate encryption schemes [15]–[17] were also presented in order to support both conjunctive keyword search and disjunctive keyword search.

As we can see, all of these multi-keyword search schemes can't support ranked search, which means the cloud server should send back the top-k most relevant files rather than all relevant files. Ranked search is highly desirable in the "pay-as-you use" cloud paradigm, because it can improve the accuracy and efficiency of query. The early ranked search schemes [18], [19] can only support single keyword search. And Cao *et al.* [20] proposed the first multi-keyword ranked search scheme (MRSE). Their scheme utilized "inner product similarity" to compute the relevant scores. However, the search time of MRSE is almost linear to the number of files in the data set, because we need to compute a relevant score on each file, even though these files don't contain any searched keyword. In order to improve the search efficiency, Sun *et al.* [21] built a tree-based structure for the whole data set and their scheme exploits vector space model with cosine measure to evaluate the similarity. Even though the search efficiency of their scheme is improved indeed, the search precision is reduced to some extent. Because the data is encrypted, updating the data set is difficult, and lots of index structure cannot support dynamics. Updating data dynamically is a challenge. At present, many researchers are devoted to the dynamic research of searchable encryption schemes. Fu *et al.* [22] presented a synonym-based multi-keyword ranked search scheme. This scheme can achieve more accurate search result and support synonym query. However, the updating cost of the above schemes is excessive. For example, if the data owner wants to add a file to her or his data set, he or she has to reconstruct the searchable index tree and all of the encrypted index vector to guarantee the cloud server work normally. To deal with this problem, Sun *et al.* [23] proposed a multi-keyword search scheme supporting dynamic operation well. However, in this scheme, the data owner needs to encrypt the internal nodes of index tree as the leaf nodes, which can lead to a high computational cost and an inefficiency of updating. So the effective multi-keyword ranked search methods with low updating cost are worth studying.

Bloom filter [24] can decide whether an element is really a member of the set easily and it has been applied to some searchable encryption schemes [25]–[27]. Bringer *et al.* [25] presented an error-tolerant searchable encryption scheme and their scheme combines Bloom filter and LSH (locality sensitive hashing) to achieve their error-tolerant aim. Kuzu *et al.* [26] utilizes Bloom filter as translation function, which maps a string into a Bloom filter, to support similarity search. By taking advantage of the space efficiency of Bloom filter, Poon *et al.* [27] proposed a phrase search scheme. And their scheme requires a low storage cost comparing with the existing phrase search schemes. However, these schemes cannot achieve multi-keyword ranked search. In this paper, we will propose a secure and effective multi-keyword ranked search scheme supporting update operations efficiently. The index tree based on Bloom filter will be designed to improve the search efficiency. And our scheme utilizes vector space model to build an index vector for every file in the outsourcing data set. The cosine similarity measure is used to compute the similarity score of one file to the search query and TF $\times$ IDF weight will be used to improve the search accuracy.

Our scheme can achieve update operation explicitly and the updating cost of our scheme is low because of the characteristics of the Bloom filter. Similarly, we will present two secure schemes to meet privacy requirements under two different threat models. The basic scheme is secure under the known ciphertext model, while the enhanced scheme is secure under the known background mode. Our contributions are summarized as follows:

(1) We design an index tree based on Bloom filter to improve the search efficiency. Our schemes can achieve the sub-linear search time. And, both the search efficiency and the index tree construction efficiency in our scheme are better than other related schemes.
(2) We propose our multi-keyword search schemes which can support dynamic operations properly and the efficiency of dynamic operations in our schemes is satisfactory.
(3) The experiments on the real-world data set show that the performance of our proposed schemes is satisfactory.

The rest of the paper is organized as follows: In Section 2, we briefly introduce the correlative definitions about system model, threat model and the design goals of our schemes. In Section 3, the notations of our schemes and preliminaries are introduced first. Then our proposed schemes are described in detail, including the basic scheme and the enhanced scheme. In Section 4, we present the security analysis of our schemes. Section 5 presents some experimental results and analysis. Our conclusions are presented in Section 6.

II. PROBLEM FORMULATION

A. SYSTEM MODEL

There are three entities, which are the data owner, the data user and the cloud server, should be considered in the system model of our proposed schemes as shown in Fig.1. The data owner is the entity such as people, companies and organizations, who wants to outsource a document collection *DC* into the cloud for the economic savings and management flexibility. But before that, the data owner must encrypt *DC* to the encrypted form *C* to ensure the confidentiality and the privacy of the data. Meanwhile, the related search index tree should also be built aiming at realizing the search capability on the encrypted data. Then the data owner will outsource the encrypted document set *C* and the corresponding search index tree to the remote cloud server. When the authorized data user wants to search the relevant documents from cloud server with the query keywords, the user must get the corresponding trapdoor *T* through search control mechanisms. After the cloud server receives the request trapdoor *T*, it will traverse the search index tree and calculate the similarity scores with the corresponding documents. Then the cloud server will return the top-*k* most relevant encrypted documents to the data user in terms of the parameter *k*, which was sent by the data user together with the search trapdoor.

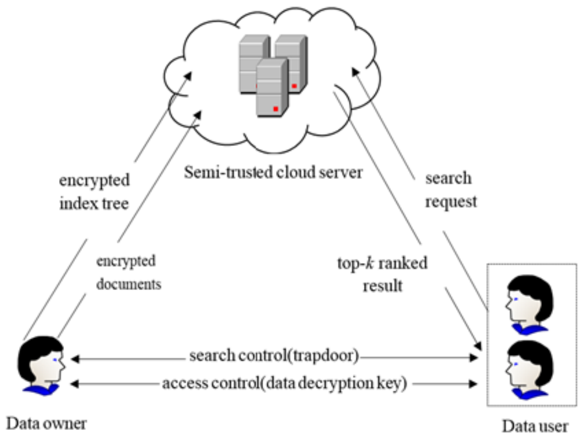


FIGURE 1. Architecture of the search over encrypted cloud data.

B. THREAT MODEL

The cloud server is considered as semi-trusted, and it may want to learn some private information about the data. So Cao *et al.* [20] proposed two threat models on the basic of what information the cloud server knows. We adapt these two threat models in our scheme.

Known ciphertext model. The cloud server knows nothing except the encrypted document collection *C*, the search index tree and the query trapdoor in this model.

Known background model. In this model, the cloud server can get access to more information, than it knows in the known ciphertext model, which may contain the interrelation of given trapdoors and the statistical information about the document set such as the TF distribution of a known keyword. So with this additional information, the server may launch the TF statistical attract to identify whether some specific keywords are in this query [28], [29].

C. DESIGN GOALS

As we have said our scheme can achieve multi-keyword ranked search supporting dynamic operation securely and effectively. So the design goals of our scheme are as follows.

- Search efficiency. The time complexity of searching should be sub-linear against the size of document set in our schemes. And the search efficiency should be superior to the current schemes.
- Dynamic. Dynamic operation contains deletion and insertion in our proposed scheme, and the efficiency of dynamic operation in our schemes should be superior to the current schemes.
- Privacy goals. Our proposed scheme can ensure the cloud server learn nothing other than the search results. So the following privacy requirements should be met in our scheme.
  (1) Index privacy. Index privacy can prevent the adversary from getting access to the plaintext information about the index. The information contains the keywords and the corresponding TF values.
  (2) Keyword privacy. The cloud server can't deduce any keyword from the query of data user, which means the trapdoors can't leak any information about the query keywords.

Trapdoor unlinkability. Given any two trapdoors, the cloud server can't determine whether they have any relationship. That is, the generated trapdoor should be randomized.

III. PROPOSED SCHEME

In this section, we will describe our proposed schemes in detail. A basic scheme will be presented first, which is secure under known ciphertext model. However, some sensitive frequency information may be leaked in the basic scheme under the stronger threat model. So, we will give an enhanced scheme later. The enhanced scheme can guarantee safety under known background model, which is stronger than known ciphertext models. The notations used in our proposed scheme are shown in Table 1.

A. PRELIMINARIES

1) VECTOR SPACE MODEL AND RANK FUNCTION

Vector space model [30] is one of the most popular similarity measure in information retrieval, which is also used extensively in multi-keyword search over encrypted cloud data. Specifically, accurate ranking search can be easily realized when the vector space model is combined with TF $\times$ IDF

TABLE 1. The notations used in our proposed schemes.

Notations	Descriptions
DC	collection of m plaintext documents, $DC = \{d_1, d_2, ..., d_m\}$
C	collection of m encrypted documents, $C = \{c_1, c_2, ..., c_m\}$
W	keyword dictionary containing n keywords, $W = \{w_1, w_2, ..., w_m\}$
I	the search index tree
D_d	the index vector of document d
BF_d	the Bloom filter of document d
W_q	the search keyword set
Q	he search index vector generated on the basic of W_q
I_d	the encrypted index vector of document d
T_{w_q}	the trapdoor for the search request

rule and similarity evaluation function. In the TF × IDF rule, TF (term frequency) is the occurrence times of the term in the corresponding document and IDF (inverse document frequency) is obtained by dividing the sum of documents in DC by the number of documents containing the term. Obviously, TF and IDF can evaluate the importance of the term from different aspects. For the vector space model combined with TF × IDF rule, each element of D_d is the normalized TF value and each element of Q is the normalized IDF value. With the index vectors of documents and the index vector of query, a similarity evaluation function can be applied to evaluate the similarity between them. There are many similarity evaluation functions in the information retrieval and we will choose a cosine measure used in [27], which is explained as follows.

N_{d,w_i} — the total occurrence times of w_i in document d.

N_{w_i} — the total number of documents containing w_i.

N—the total number of keywords in the keyword dictionary W.

M—the total number of documents in the document set DC.

TF_{d,w_i} — the TF value of w_i in document d.

IDF_{w_i} — the IDF value of w_i in the document set DC.

With these notations, the similarity evaluation function is defined as:

$$SC(Q, d) = \frac{\sum_{i=1}^{N} TF_{d,w_i} \cdot IDF_{w_i}}{\sqrt{\sum_{j=1}^{N} (TF_{d,w_i})^2} \cdot \sqrt{\sum_{j=1}^{N} (IDF_{w_i})^2}} \quad (1)$$

where $TF_{d,w_i} = 1 + \ln N_{d,w_i}$ and $IDF_{w_i} = \ln(1 + \frac{M}{N_{w_i}})$. In Formula (1), $\frac{TF_{d,w_i}}{\sqrt{\sum_{j=1}^{N} (TF_{d,w_i})^2}}$ is called as the normalized TF, and $\frac{IDF_{w_i}}{\sqrt{\sum_{j=1}^{N} \sum_{j=1}^{N} (IDF_{w_i})^2}}$ is called as the normalized IDF.

2) BLOOM FILTER

Bloom filter is a space-efficient data structure, which is used to decide whether an element is really the member of the set. Assume that there is a set $\boldsymbol{S} = \{\boldsymbol{x_1}, \boldsymbol{x_2}, \cdots, \boldsymbol{x_n}\}$, and the set S can be represented as a Bloom filter, which is an array of b bits initialized with 0. Generally, the generating algorithm of Bloom filter utilizes r independent hash functions $\boldsymbol{h_i}(i = 1, 2, \cdots, \boldsymbol{r})$, where $\boldsymbol{h_i} : \{\boldsymbol{0}, \boldsymbol{1}\}^* \rightarrow [\boldsymbol{1}, \boldsymbol{b}]$. With the hash functions, every element x can be mapped to r random numbers $\boldsymbol{h_1}(\boldsymbol{x}), \boldsymbol{h_2}(\boldsymbol{x}), \cdots, \boldsymbol{h_r}(\boldsymbol{x})$ by computing hi(x) and the corresponding bits at this positions should be set to 1. When we want to test whether the element x is contained in the set S, we just only check whether the bits at the positions $\boldsymbol{h_1}(\boldsymbol{x}), \boldsymbol{h_2}(\boldsymbol{x}), \cdots, \boldsymbol{h_r}(\boldsymbol{x})$ are equal to 1. If any of the bits at these positions is 0, the element is definitely not in the set. Otherwise, x is considered to be in the set. However, there may occur a positive false, which means the element x is decided to be in the set, but it's not in fact. Fortunately, if the parameters and hash functions are set properly, the positive false rate can be negligible.

As we have said, our schemes utilize Bloom filter to build the search index tree, so we will discuss its correctness and security in the following sections.

3) SEARCH INDEX TREE

The search index tree in our scheme is a balance binary tree, which can improve search efficiency and support dynamic operation with low cost. In order to achieve these design goals, the data structure of our search index tree node u is defined as $\{\boldsymbol{FID}, \boldsymbol{D_u}, \boldsymbol{BF_u}, \boldsymbol{l}, \boldsymbol{r}\}$, where l and r represent the left child and right child of u, respectively. Every leaf node corresponds to a specific document in our search index tree. So, if there are m documents in total, we need to construct a search index tree with m leaf nodes. As to a leaf node corresponding to the document d, FID denotes the identifier of d, $\boldsymbol{D_u}$ is the index vector of d and $\boldsymbol{BF_u}$ represents the Bloom filter of d where each keyword of d is mapped to r random positions. With these leaf nodes, the search index tree can be built by performing a postorder traversal. For every internal node u including root node, the FID and $\boldsymbol{D_u}$ can be set to null and $\boldsymbol{BF_u}[\boldsymbol{i}] = \boldsymbol{BF_l}[\boldsymbol{i}]$ ***or*** $\boldsymbol{BF_r}[\boldsymbol{i}]$. Namely, $\boldsymbol{BF_u}[\boldsymbol{i}]$ should be set to 0 if and only if both $\boldsymbol{BF_l}[\boldsymbol{i}]$ and $\boldsymbol{BF_r}[\boldsymbol{i}]$ are 0. Otherwise, $\boldsymbol{BF_u}[\boldsymbol{i}]$ should be set to 1. The detail algorithm of index tree construction is shown in Algorithm 2.

An example of our proposed search index tree is shown as Fig.2. In this example, we set b = 10, r = 2. The document $\boldsymbol{d_1}$ contains $\boldsymbol{w_1}, \boldsymbol{w_4}$, $\boldsymbol{d_2}$ contains $\boldsymbol{w_1}, \boldsymbol{w_3}, \boldsymbol{w_4}$, $\boldsymbol{d_3}$ contains $\boldsymbol{w_2}, \boldsymbol{w_5}$ and $\boldsymbol{d_4}$ contains $\mathbf{w_3}$. Then, the plaintext search index tree can be built by BuildIndexTree() in Algorithm 1. In addition, the search algorithm to find the relevant leaf nodes is explained in section 3.

B. THE BASIC SCHEME

Our basic scheme can achieve privacy-preserving multi-keyword ranked search supporting dynamic operation under

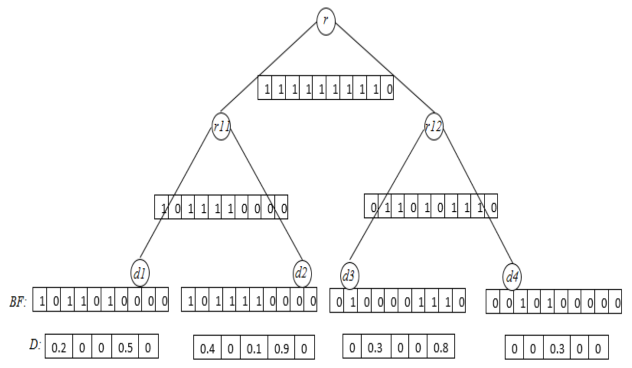


FIGURE 2. Search index tree for the document set $DC = \{d_1, d_2, d_3, d_4\}$ with the keyword dictionary $W = \{w_1, w_2, \ldots, w_5\}$. And the Bloom filter generation algorithm utilizes two private key $\{K_1, K_2\}$ as $h_{K_1}(w_1) = 1$, $h_{K_1}(w_2) = 7$, $h_{K_1}(w_3) = 3$, $h_{K_1}(w_4) = 4$, $h_{K_1}(w_5) = 8$ and $h_{K_2}(w_1) = 3$, $h_{K_2}(w_2) = 10$, $h_{K_2}(w_3) = 5$, $h_{K_2}(w_4) = 6$, $h_{K_2}(w_5) = 9$.

the known ciphertext model. For convenience, we assume the data owner wants to outsource her or his document set $DC = \{d_1, d_2, \ldots, d_m\}$ to the cloud server. Before that, he or she must encrypt DC for the consideration of privacy. And a secure index I must be built based on the extracted keyword dictionary $\boldsymbol{W} = \{\boldsymbol{w_1}, \boldsymbol{w_2}, \cdots, \boldsymbol{w_n}\}$ for the efficient searchable capability. Then, the following steps will be performed to achieve the design goals for our scheme.

- Setup

Firstly, the data owner decides the length of Bloom filter b bits and the value of r. And r represents the number of positions that a keyword should be mapped to. Then, the data owner generates a 4-tuple $\{\boldsymbol{S}, \boldsymbol{M_1}, \boldsymbol{M_2}, \boldsymbol{K_h}\}$ as the secret key SK. Here, S is a random n-bits vector, $\{\boldsymbol{M_1}, \boldsymbol{M_2}\}$ are two $\boldsymbol{n} \times \boldsymbol{n}$ invertible matrices and $\boldsymbol{K_h}$ is composed of r private keys $\{\boldsymbol{K_1}, \boldsymbol{K_2}, \cdots, \boldsymbol{K_r}\}$ utilized to produce the Bloom filter.

- GenIndex(DC, SK)

In this phase, the data owner will encrypt the outsourcing documents and build the encrypt index tree as the following steps:

Step 1: The data owner calls BuildIndexTree(DC, $\boldsymbol{K_h}$, b) as shown in Algorithm 2, to generate an unencrypted search index tree. To obfuscate the Bloom filter further, p dummy keywords are added during the generation of Bloom filter for each document in the step 1 of BuildIndexTree(DC, $\boldsymbol{K_h}$, b). Or more specifically, $\boldsymbol{r} \times \boldsymbol{p}$ randomly selected positions of the array are set to 1.

Step 2: The data owner encrypts the index vector $\boldsymbol{D_d}$ of every leaf node in the search index tree using the secret key SK. Specifically, if $\boldsymbol{S}[\boldsymbol{i}]$ is equal to 0, $\boldsymbol{D'_d}[\boldsymbol{i}]$ and $\boldsymbol{D''_d}[\boldsymbol{i}]$ are set to $\boldsymbol{D_d}[\boldsymbol{i}]$; if $\boldsymbol{S}[\boldsymbol{i}]$ is equal to 1, $\boldsymbol{D'_d}[\boldsymbol{i}]$ and $\boldsymbol{D''_d}[\boldsymbol{i}]$ are set to two random numbers as long as their sum is $\boldsymbol{D_d}[\boldsymbol{i}]$, which can be expressed as:

$$\begin{cases} \boldsymbol{D'}[i] = \boldsymbol{D''}[i] = \boldsymbol{D}[i], \boldsymbol{ifS}[i] = 0 \\ \boldsymbol{D'}[i]\,\mathcal{C}\,\boldsymbol{D''}[i] = \boldsymbol{D}[i], \boldsymbol{ifS}[i] = 1 \end{cases} \quad (2)$$

Algorithm 1 BuildIndexTree(DC, $\mathbf{K_h}$, b)

```
1: For each document $d_i(i= 1, 2, \cdots, m)$ in the document set, generate a corresponding leaf node for it as following procedures:
  • Generate a unique identifier for $d_i$ and set this identifier as the FID of this leaf node.
  • Generate the index vector $D_{d_i}$ according to the keyword dictionary $W= \{w_1, w_2, \cdots, w_n\}$, where the length of $D_{d_i}$ is the size of keyword dictionary and each dimension $D_{d_i}[j]$ is the normalized of TF value of $w_j$ in the document $d_i$.
  • Generate a Bloom filter $BF_{d_i}$ for $d_i$. Assume the generating algorithm of Bloom filter utilize hash functions $h_{sk}$, where $h_{sk}:\{0,1\}^* \rightarrow [1, b]$. So for every keyword w in the document $d_i$, the positions of $BF_{d_i}$ at $h_{K_1}(w), h_{K_2}(w), \cdots, h_{K_r}(w)$ will be set to 1.
2: Generate a search index tree I whose leaf nodes are the m nodes generated in the above step. The $FID$, $D_u$ and $BF_u$ of the internal node u are set to null initially.
3: Update the Bloom filter of every internal node. For each internal node u, $BF_u$ can be computed according to the Bloom filters of its left and right child node recursively. More specifically, $BF_u[i] = BF_l[i]$ or $BF_r[i]$, where "or" is the Boolean OR operator.
4: Output the search index tree I.
```

Algorithm 2 SearchIndexTree ($\mathbf{BF_q}$, **IndexTreeNode u**)

```
1. if $u$ is an internal node then
2. Initialize count to 0;
      for $i = 1$ to the length of $BF_q$ do
         if($BF_q[i] == 1$ and $BF_u[i] == 1$) then
            count ++;
3. end if
      end for
      if (count >= $\lceil r/a \rceil$) then
         SearchIndexTree ($BF_q$, $u.l$);
         SearchIndexTree ($BF_q$, $u.r$);
      end if
   else
4. return the current node $u$;
5. end if
```

$\boldsymbol{I_d}$ can be generated as $\{\boldsymbol{M_1^T D'_d}, \boldsymbol{M_2^T D''_d}\}$ and stored in the corresponding leaf node by replacing $\boldsymbol{D_d}$.

Step 3: The data owner removes the Bloom filter of all leaf node before uploading the encrypted document set C and encrypted index tree to the cloud server.

- GenQuery ($\boldsymbol{W_q}$)

To avoid the leakage of privacy, the data user will compute a trapdoor according to the search keywords set, which also can be regarded as the encrypted form of a search request. The detail procedure is shown as follows:

Step1: The data user inputs her or his query keyword set W_q and computes a query vector Q, the length of which is the same as the index vector of document. And $Q[i]$ is set to the normalized IDF value, if $w_i \in W_q$. Otherwise, $Q[i]$ is set to 0.

Step2: The data user encrypts Q to generate the trapdoor T_{W_q} by using the secret key SK. The same method is utilized to spilt Q into $\{Q', Q''\}$. But the difference is that if $S[i]$ is equal to 0, $Q'[i]$ and $Q''[i]$ are set to two random numbers as long as their sum is $Q[i]$; if $S[i]$ is equal to 1, $Q'[i]$ and $Q''[i]$ are set to the same as $Q[i]$, which can be expressed as:

$$\begin{cases} Q'[i] + Q''[i] = Q[i], \text{if} S[i] = 0 \\ Q'[i] = Q''[i] = Q[i], \text{if} S[i] = 1 \end{cases} \tag{3}$$

And the trapdoor T_{W_q} can be generated as $\{M_1^{-1}Q', M_2^{-1}Q''\}$.

Step3: The data user computes the query Bloom filter BF_q to improve the search efficiency. For each search keyword w $\in W_q$, r random numbers $h_{K_1}(w), h_{K_2}(w), \cdots, h_{K_r}(w)$ are generated by utilizing the same procedure in BuildIndexTree(DC, K_h, b). However, only $\lceil r/a \rceil$, $(1 \leq a \leq r)$ positions, which are selected randomly from $h_{K_1}(w), h_{K_2}(w), \cdots, h_{K_r}(w)$, are set to 1 in BF_q. In general, the value of a is less than 2 aimed to guarantee efficiency of our scheme, which will be discussed in the following sections. After T_{W_q} and BF_q are generated, both will be sent to the cloud server.

- Search (T_{W_q}, I)

With the query Bloom filter BF_q, the cloud server calls **w**ith the query Bloom filter $\boldsymbol{BF_q}$, the cloud server calls the following Algorithm 2 SearchIndexTree ($\boldsymbol{BF_q, I.root}$) to get all the relevant leaf nodes. And the similarity score for every relevant leaf node u and trapdoor $\boldsymbol{T_{W_q}}$ can be computed by the following formula:

$$\begin{aligned} SC(I_u, T_{w_q}) &= \{M_1^T D'_u, M_2^T D''_u\} \cdot \{M_1^{-1}Q', M_2^{-1}Q''\} \\ &= D'_u \cdot Q' + D''_u \cdot Q'' \\ &= D_u \cdot Q \end{aligned} \tag{4}$$

With the similarity scores of all the relevant documents, the cloud server sorts the scores utilizing appropriate sorting algorithm and returns the top-k encrypted documents to the data user.

C. THE ENHANCED SCHEME

As formula (4) shows, the similarity score calculated from I_u and T_{w_q} is the same as that calculated from D_u and Q. That is, two different trapdoors generated from the same query keywords will have the same similarity score with the same document. However, this will incur privacy leakage in the background model. So we must break this equality to improve the security of our scheme. We can introduce randomness to I_u and T_{w_q} during their generations. Specifically, the index vector will be extended to $(n + U)$ dimensions, where U is the number of dummy keywords in our enhanced scheme.

Our enhanced scheme is almost the same as the basic scheme, and the changed details are that:

1) when the secret key SK is generated, the length of S becomes $(n+U)$ bits and $\{M_1, M_2\}$ are changed to two $(n + U) \times (n + U)$ matrices;
2) when generating the index vector D_d in the step 2 of BuildIndexTree(DC, K_h, b), D_d is also extended to $(n+U)$ dimensions and the extended dimensions $D_d[n+j]$, $(j = 1, 2, \ldots, U)$ is set to an random number ε_j;
3) the query vector is extended to $(n+U)$ dimensions and V positions out of extended U dimensions, which is randomly selected, are set to 1 before the encryption procedure;
4) the similarity of I_u and T_{w_q} will be $(D_u \cdot Q + \sum \varepsilon_j^{(V)})$ rather than $D_u \cdot Q$ **,** Which makes $Q[n+j] = 1$

D. THE UPDATE OPERATION

Our scheme can achieve an efficient, multi-keyword search and support dynamic operation, such as the insertion or deletion of a document. We utilize Bloom filter to construct the corresponding search index tree. When a document needs to be inserted or deleted, we just modify the structure of search index tree slightly. Specifically, only the nodes on the path from the changed leaf node to the root need to be updated. In addition, an unencrypted search index tree also needs to be stored on the data owner side for the reduction of communication overhead. The dynamic operation process is presented as follows:

- GenUpdateInfo(*SK, I, doc, UT*)

In this phase, the data owner will generate the changed path information P, which is a subtree containing all nodes on it. For example, if we want to delete the document d_4 **d4** in Fig.2, P is composed of $\{r, r_{12}\}$ Here, the parameter *SK* is the secret key generated in the setup phase, I is the search index tree, and UT represents the type of dynamic operation. So, two situations will be considered according to UT.

1) If *UT* represents deletion of a document, the data owner will find the corresponding leaf node and delete this node. However, if deletion of the node breaks the balance structure of the search index tree, the data owner can set this leaf node as a blank node rather than removing it. In the blank node, the *FID* is set to null, the encrypted index vector and bloom filter are set to 0. After that, the data owner updates the Bloom filter of the changed subtree and stores the changed path information into *P*.
2) If *UT* represents insertion of a new document, a new leaf node will be generated as the process of step1 in algorithm BuildIndexTree(DC, K_h, b). With the leaf node, the data owner inserts it into the index tree I as a leaf node without breaking the balance structure of the search index tree. And the blank node will be considered firstly, which can be found by the blank node list. Then, the data owner updates the Bloom filter of the nodes on the path from root to the new leaf node as $BF_u[i] = BF_l[i] or BF_r[i]$. Then, the data owner stores the changed path information into P, which will be sent to the cloud server along with the encrypted form of *doc*. In addition, while finding the proper location in I to insert the new leaf node, some blank nodes are

usually preferred, which are generated from the process of the deletion operation.

- Update(I, P, UT)

In this phase, the cloud server will update the search index tree that stored in it. Specifically, the cloud server updates the corresponding subtree of I according to P. And if UT represents insertion of a new document, the cloud server adds the encrypted form of the new document to C. Otherwise, it deletes the corresponding encrypted document from C.

The IDF values of keywords are utilized to generate query trapdoor. However, some values may be changed after dynamic operations. So the data owner must update the changed IDF values in time. In fact, the data owner does not have to update the IDF values every time when dynamic operations happen. Because the IDF values change little after several times of documents updating.

As for modifying a document, such as the deletion or insertion of some keywords, this operation can be completed by combinations of the above two operations. Specifically, we can delete the original document in the cloud server, firstly. Then, we insert the new document, which have been modified. In addition, the size of keyword dictionary W may be changed when dynamic operations occur, because there may be new keywords in the new documents. In the most existing schemes, the data owner has to rebuild the search index tree, which is time-consuming. However, our schemes do not have to rebuild the search index tree. For the Bloom Filter, we can just map the new keyword to r positions and the effect of its length will be discussed in Section 4. For the index vector, the length is determined by the size of the keyword dictionary. We can preserve some blank entries in the keyword dictionary and the corresponding positions of index vector are set to 0. So the new keywords can be inserted by replacing the blank entries when the data owner adds some new documents.

IV. SECURITY ANALYSIS

In this section, we present the security analysis of our schemes. And we will analyze our schemes from the privacy requirements described in Section 2.3.

A. SECURITY ANALYSIS OF OUR BASIC SCHEME

Our basic scheme is a privacy–preserving multi-ranked search scheme in the ciphertext model. And we analyze that it can meet the privacy requirements as follows.

1) Index privacy

In our basic scheme, the index privacy involves the encrypted index vector I_d and the Bloom filter BF_d. For I_d, it was obfuscated by secret key SK and the cloud server can't get the original D_d without SK. For BF_d, we utilize r independent hash functions to map a keyword to r positions to BF_d. So BF_d does not reveal keywords information. Besides, we add dummy keywords to BF_d to obfuscate it during the generation of Bloom filter for better privacy protection. Therefore, our basic scheme satisfies the requirements of index privacy.

2) Trapdoor unlinkability

During the generation of the trapdoor, some dimensions of the search index vector may be spilt into two random numbers. So, the same search index vector will be encrypted to different trapdoors. For the query Bloom filter BF_q, we only randomly selected $\lceil r/a \rceil$ positions for a query word and set the corresponding positions of BF_q to 1. Thus, the same query keyword set will be generated different Bloom filters. Therefore, our basic scheme satisfies the requirements of trapdoor unlinkability in the known ciphertext model.

3) Keyword privacy

The above analysis shows that the cloud server can't get any information about keywords from the indexes and trapdoors without more information. And the cloud server can't infer useful information either in the known ciphertext model. So, our basic scheme can guarantee keyword privacy in the known ciphertext model. However, the cloud sever will have more knowledge in the known background model such as the TF distribution of keywords. Thus, the cloud server may deduce the keyword information by analyzing the TF distribution. Therefore, our basic scheme may leak the keyword information in the known background model.

B. SECURITY ANALYSIS OF OUR ENHANCED SCHEME

In our scheme, the cloud server should use the bloom filter to search the top-k results. Over the search process, the server needs to calculate the relevance scores for each file. It may expose some information about the data, but the data sets are encrypted, and we can get different ciphertext for one file when we encrypt it more time. So the server cannot know the detailed information about the top-k list files. Our scheme is secure under known background model.

We analyze the security of our enhanced scheme similar to that in our basic scheme.

1) Index privacy

The encrypted method of index vector and the generation of Bloom filter in the enhanced scheme are the same as that in the basic scheme. So the enhanced scheme can guarantee index privacy in the known background model.

2) Trapdoor unlinkability

Inherited from our basic scheme, the same search index vector will be encrypted to different trapdoors. Besides, our enhanced scheme breaks the equality between the final similarity scores and $(D_u \cdot Q)$ because of the randomness of $\sum \varepsilon_j^{(V)}$. Thus, the similarity scores of the same query will be different. So the enhanced scheme satisfies the requirement of trapdoor unlinkability in the known background model.

3) Keyword privacy

It's difficult for the cloud server to collect the specific TF distributions of keywords because the final similarity score is obfuscated by the random $\sum \varepsilon_j^{(V)}$. And the cloud server cannot identify whether some keyword is in the query without TF

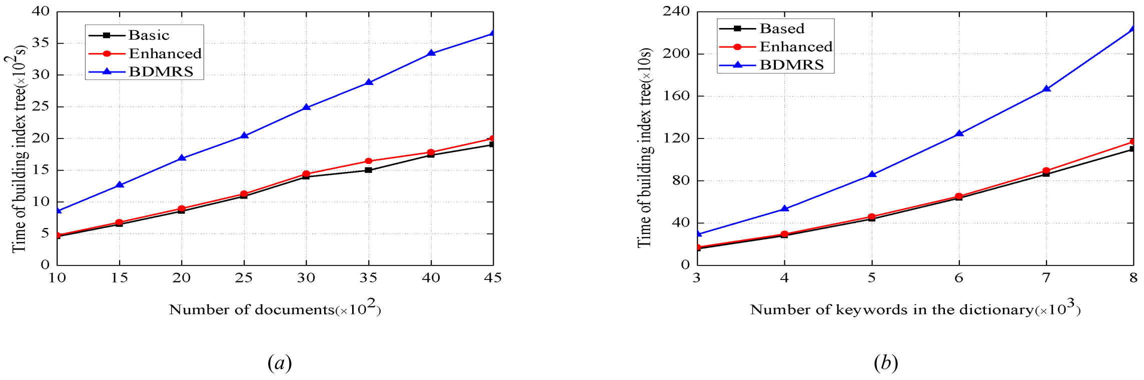


FIGURE 3. Time costs of building index tree: (a) different size of document sets with the same dictionary, $n = 5000$; (b) the same document set with a different size keyword dictionary, $m = 1000$.

distributions. As it was analyzed in [20], the keyword privacy can be protected in our enhanced scheme if the parameters, such as U and V, are selected properly.

However, the dummy keywords can affect the accuracy of search and ε_j must be chose properly. Assume every ε_j is set to follow the same uniform distribution $M(\mu^{'} - \text{c}, \mu^{'} + \text{c})$. Then, $\sum \varepsilon_j^{(V)}$ will follow the Normal distribution $N(\mu, \sigma^2)$ if the values of $\mu^{'}$ and c are set to μ/V and $\sqrt{3/V} * \sigma$ on the basic of the central limit theorem, recespectively. Thus, the value of σ can be set as a system parameter to make a trade-off between security and accuracy.

V. PERFORMANCE ANALYSIS

In this section, we present a detailed analysis for the performance of our proposed schemes, and we implemented the experiment by using JAVA language on a Linux Server with Intel 2.9 GHz Processor. The real dataset of National Science Foundation Research Awards Abstracts 1990-2003 [31] is used in our experiment. Specifically, we randomly select different numbers of documents to build the experimental dataset.

A. EFFICIENCY

1) CONSTRUCTION FOR INDEX TREE

Many schemes, such as Sun *et al.* [23], utilize binary search tree to improve the search efficiency of their schemes. And for the consideration of privacy, the internal nodes of the tree must be encrypted, which is a time-consuming operation. We also build a search

Index tree for our scheme. However, our scheme need not encrypt the internal nodes because of the property of the Bloom filter. So the index tree construction time of our scheme will be reduced significantly. Specifically, the main time cost process of the index tree construction contains the construction of unencrypted search index tree and the encryption of the index vector for each leaf node. For the construction of unencrypted index tree, the index vector and the Bloom filter generations for each document are the main considerable operations, which is related to the size of document set. As to the encryption operation of every leaf node, two multiplication operations of a $n \times n$ invertible matrice and a n-dimension vector mainly should be considered, which takes $O(n^2)$ time in our basic scheme and takes $O((n + U)^2)$ time in our enhanced scheme. So the size of document set m and the number of keywords n are the two main factors for the construction of index tree.

We compare the time costs required to build an index tree of our scheme with those of BDMRS in [23]. Fig. 3(a) demonstrates that the time costs of building index tree is nearly linearly related to the number of documents when the size of the keyword set is fixed. And the time costs of our enhanced scheme is a little more than our basic scheme because of the insertion of the dummy keywords. Fig. 3(b) shows that the size of the keyword dictionary can have an enormous impact on the time cost of building the index tree, which increases with the square of the number of keywords in the dictionary. Obviously, the main influencing factor for computational cost is the encryption of the leaf nodes, which involves the multiplications of matrices. Fig.3 also shows that BDMRS takes almost twice the time of our scheme to build the encrypted index tree. Even though the index tree construction is a one-time operation, the improvement of our schemes also is significant.

2) TRAPDOOR GENERATION

Our schemes utilize the similar encryption method to generate the trapdoor, which involves the spilt operation of the query vector and two multiplications of the $n \times n$ invertible matrices. So the time cost of trapdoor generation depends mainly on the number of keywords. Fig. 4(a) shows that the time cost grows exponentially with the increment of the number of keywords in the dictionary. Fig. 4(b) shows that the number of keywords in the query request has little influence on the time cost of generating the trapdoor. This feature can benefit our multi-keyword search scheme. And the time cost of our basic scheme is a little lower than that of our enhanced scheme because of the insertion of dummy keywords in the enhanced scheme.

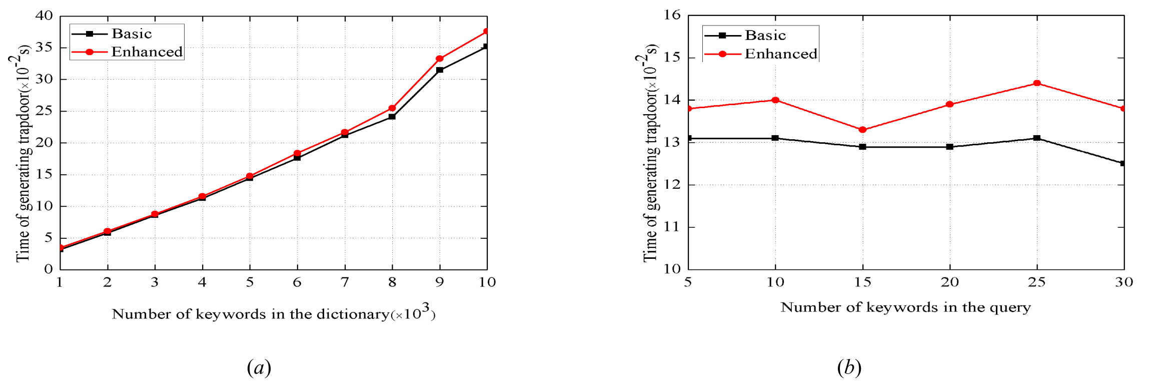


FIGURE 4. **Time cost of generating trapdoor: (a) the same 10 query keywords within different size of keyword dictionary; (b) different number of query keywords within the same keywords dictionary, $n = 5000$.**

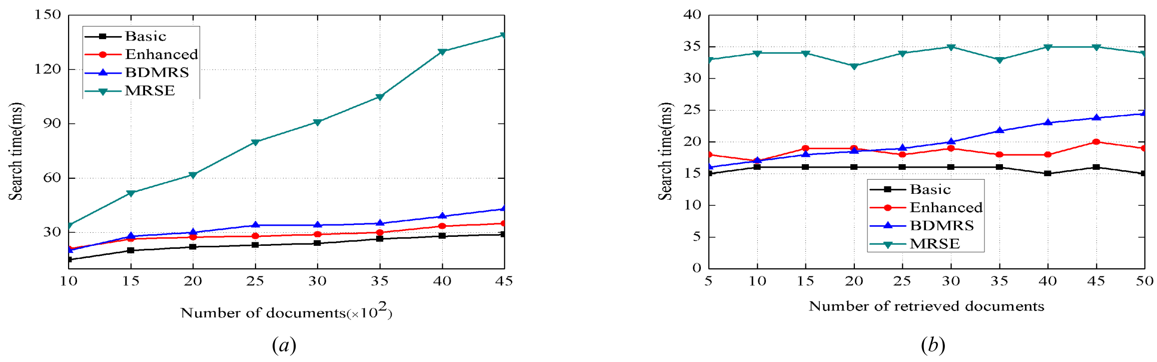


FIGURE 5. **Search efficiency with the same 10 keywords as input: (a) different size of document sets with the same size of keywords dictionary, $n = 5000$; (b) different number of retrieved documents with the same document set and keyword dictionary.**

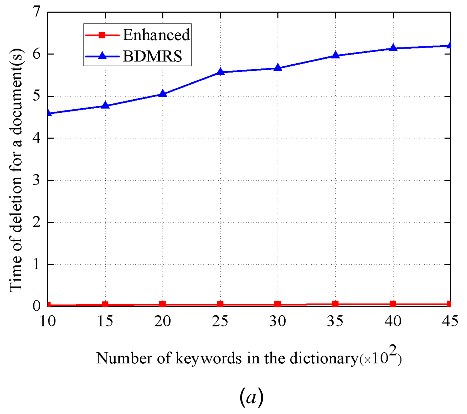
3) QUERY

The cloud server should consider the documents that contain the searched keywords rather than all of the documents when it retrieves the top-$k$documents. And according to the characteristics of Bloom filter, we can easily decide whether an element is really a member of the set. So, we utilize the Bloom filter to build a search index tree. As to the search algorithm, we can see that our schemes first traverse the index tree to determine the relevant leaf nodes that contain any of the search keywords. Then the cloud server computes the similarity scores between those leaf nodes and the request trapdoor. Thus, only the leaf nodes containing the search keywords will be traversed rather than all leaf nodes. Since our search index tree is a balanced binary tree, its maximum height is about $logm$. If we assume that there are t leaf nodes that contain any of the search keywords, the complexity of finding the relevant leaf nodes is $O(tlogm)$, and the complexity of the similarity calculation is $O(tn)$. So, the search time is $O(tlogm + tn)$.

Fig. 5(a) shows that our schemes have a logarithmic search time that is almost consistent with the size of the document set when the size of the keywords dictionary is fixed. And the search time of our schemes is far lower than that of MRSE in [20], which has a search time that is linearly related to the number of documents. Fig. 5(a) also demonstrates that, even though BDMRS utilizes "Greedy Depth-first Traverse Strategy" to improve the search efficiency, our schemes have the similar performance as BDMRS. Fig. 5(b) shows that the value of k doesn't affect the search time of our schemes and MRSE. However, the search time of BDMRS may increase somewhat as the number of documents retrieved increases.

4) UPDATE

Some data owners often update their outsourcing data except for retrieving them, so dynamic operation is a significant service. As we have explained, our scheme can support dynamic operation well, which means deletion or addition of a document cost little time and there is no privacy leakage. Because the dynamic operation of our scheme is similarity to that of Sun et al. [23], only the nodes, which are in the path from the root to the related leaf node, will be involved when the dynamic operations occur. However, their scheme must encrypt the internal node by the multiplications of the $n \times n$ invertible matrices, which can take lots of time if the index tree is very high. Unlike that, our scheme utilizes the Bloom filter to avoid the encryption of the internal nodes and the generation of the Bloom filter just takes a little time. So, the efficiency of dynamic operation in our scheme is satisfactory. To demonstrate that, we compared our scheme with BDMRS



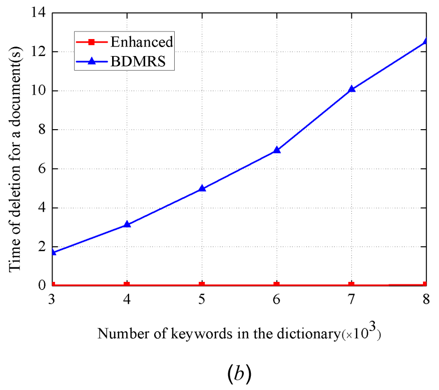


FIGURE 6. Time cost for deletion of a document: (a) different size document sets with the same size keywords dictionary, $n = 5000$; (b) same document sets with different sizes of the keyword dictionary, $m = 1000$.

in Fig. 6, when executing the deletion of a document. As the cost of deletion in our basic scheme is the same as that in our enhanced scheme, we only show the performance of the enhanced scheme. Fig. 6(a) shows that both our scheme and BDMRS almost take logarithmic time with the number of documents when the size of the keyword dictionary is fixed. However, the time cost of BDMRS is much greater than that of our scheme. For example, when there are 3000 documents, BDMRS requires 5,665ms, whereas our enhanced scheme only requires 50ms. Fig. 6(b) shows that the time cost of our scheme has almost no relationship with the size of the keyword dictionary. On the contrary, the cost time of BDMRS increases exponentially as the size of the dictionary increases. So, our schemes can deal with the dynamic operation more efficiently.

B. PRECISION AND PRIVACY

Our enhanced scheme utilizes dummy keywords to satisfy the requirements of keyword privacy under the background model. But it will influence the accuracy of the search results. In order to quantify the influence, we use "precision definition" in [20] to measure the quantity of the influence. And it is defined as $P_K = {k'}/{k}$, where k' is the number of real top-k documents returned by the cloud server. Fig. 7(a) shows that a smaller σ causes the scheme to have higher precision. So, it seems that we can utilize a smaller σ to guarantee the precision of our scheme. However, the value of σ also influences "rank privacy", which also is proposed by Cao *et al.* [20]. The "rank privacy" can be computed as $\tilde{P}_k = \sum \tilde{p}_d / k^2$. Here, $\tilde{p}_d$ for the document d denotes $|r_d - \tilde{r}_d|$, where r_d is the rank number of d in the retrieved top-k documents, and $\tilde{r}_d$ is its rank number in the real ranked documents. On the contrary, Fig. 7(b) shows that a higher σ can obtain a larger rank privacy, which means the rank information can be protected better by our scheme. This is, the standard deviation σ can affect the precision and the rank privacy at the same time. So σ can be treated as a system parameter, which can make a tradeoff between search precision and rank privacy.

Besides, our schemes utilize the search index tree based on the Bloom filter to determine the relevant documents, and the Bloom filter allows false positives. So it seems that the search index tree may affect the 'precision' of our schemes. However, it will not occur, because even a leaf node, which doesn't contain the searched keywords, can be returned by performing the algorithm SearchIndexTree (BF_q, $IndexTreeNode\ u$), and the corresponding document also can be excluded because the similarity score between this document and the request trapdoor is nearly 0. What's more, the probability of a false positive is very small, which can be neglected, if the parameters of Bloom filter are chosen properly. More specifically, assume that the r hash functions are completely random, then the probability of one bit is 0 in the Bloom filter is $(1 - 1/b)^{r \cdot n}$, which is approximately equal to $e^{-r \cdot n/b}$ when b is sufficiently large. So the probability of a false positive P in our schemes is expressed as:

$$P = (1 - e^{-r \cdot n/b})^{\lceil r/a \rceil} \tag{5}$$

In our experiments, we set $b = 65535$, $r = 6$, and $a = 2/3$. P can be computed from formula (5), and the values are shown in Table 2. We can see that even though the value of P increases with the increasements of n, P is still small enough to be neglected.

TABLE 2. The probability of a false positive.

n	P
1000	0.000059
1500	0.000271
2000	0.000784
2500	0.001751
3000	0.003327
3500	0.005650
4000	0.013001

We will consider how to choose a proper r when the length of Bloom filter b and the size of keyword set n are fixed. With the knowledge of derivative, we can know that if, and only if, $e^{-r \cdot n/b}$ is equal to $1/2$, which means $r = b \cdot ln2/n$, P

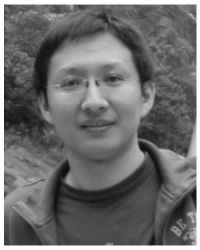
is minimum, and the minimum value is $(1/2)^{\lceil r/a \rceil}$. So, if r and a are chosen properly, we can minimize the value of P while making our scheme satisfy the requirement of trapdoor unlinkability.

VI. CONCLUSION

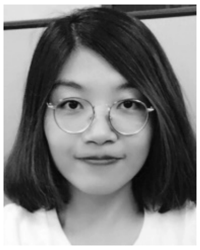
In this paper, we proposed a secure and effective, multi-keyword, ranked search scheme over encrypted cloud data. Also, our scheme more efficiently supports dynamic operations that contain deletions or insertions in a document. To perform a multi-keyword ranked search, our scheme utilizes the vector space model combined with the TF × IDF rule and the cosine similarity measure to evaluate the similarity between the documents and the query request. To improve the efficiency of the search, a search index tree based on the Bloom filter is built to determine the relevant documents. In addition, the search index tree also can reduce the cost of dynamic operations because of the properties of the Bloom filter. Finally, the experimental results show that our scheme can achieve the design goals efficiently and effectively.

REFERENCES

[1] L. M. Vaquero, L. Rodero-Merino, J. Caceres, and M. Lindner, "A break in the clouds: Towards a cloud definition," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 39, no. 1, pp. 50–55, 2008.
[2] D. X. Song, D. Wagner, and A. Perrig, "Practical techniques for searches on encrypted data," in *Proc. IEEE Symp. Secur. Privacy*, May 2000, pp. 44–55.
[3] Z. Xia, X. Wang, X. Sun, and Q. Wang, "A secure and dynamic multi-keyword ranked search scheme over encrypted cloud data," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 2, pp. 340–352, Jan. 2016.
[4] R. Curtmola, J. A. Garay, S. Kamara, and R. Ostrovsky, "Searchable symmetric encryption: Improved definitions and efficient constructions," in *Proc. 13th ACM Conf. Comput. Commun. Secur. (CCS)*, 2006, vol. 19, no. 5, pp. 79–88.
[5] D. Boneh, G. Di Crescenzo, R. Ostrovsky, and G. Persiano, "Public key encryption with keyword search," in *Advances in Cryptology-EUROCRYPT*. Berlin, Germany: Springer, 2004, pp. 506–522.
[6] Z. Fu, X. Sun, Q. Liu, L. Zhou, and J. Shu, "Achieving efficient cloud search services: Multi-keyword ranked search over encrypted cloud data supporting parallel computing," *IEICE Trans. Commun.*, vol. E98-B, no. 1, pp. 190–200, 2015.
[7] J. Li, Q. Wang, C. Wang, N. Cao, K. Ren, and W. Lou, "Fuzzy keyword search over encrypted data in cloud computing," in *Proc. IEEE INFOCOM*, San Diego, CA, USA, Mar. 2010, pp. 1–5.
[8] P. van Liesdonk, S. Sedghi, J. Doumen, P. Hartel, and W. Jonker, "Computationally efficient searchable symmetric encryption," in *Proc. Workshop Secure Data Manage. (SDM)*, 2010, pp. 87–100.
[9] Z. Fu, K. Ren, J. Shu, X. Sun, and F. Huang, "Enabling personalized search over encrypted outsourced data with efficiency improvement," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 9, pp. 2546–2559, Sep. 2016.
[10] S. Kamara, C. Papamanthou, and T. Roeder, "Dynamic searchable symmetric encryption," in *Proc. ACM Conf. Comput. Commun. Secur.*, 2012, pp. 965–976.
[11] L. Ballard, S. Kamara, and F. Monrose, "Achieving efficient conjunctive keyword searches over encrypted data," in *Proc. 7th Int. Conf. Inf. Commun. Secur.* Beijing, China: Springer-Verlag, Dec. 2005, pp. 414–426.
[12] Y. H. Hwang and P. J. Lee, "Public key encryption with conjunctive keyword search and its extension to a multi-user system," in *Proc. 1st Int. Conf. Pairing-Based Cryptogr.* Tokyo, Japan: Springer-Verlag, Jul. 2007, pp. 2–22.
[13] D. Boneh and B. Waters, "Conjunctive, subset, and range queries on encrypted data," in *Proc. Theory Cryptogr. Conf. (TCC)*, 2006, pp. 535–554.
[14] B. Zhang and F. Zhang, "An efficient public key encryption with conjunctive-subset keywords search," *J. Netw. Comput. Appl.*, vol. 34, no. 1, pp. 262–267, 2011.
[15] J. Katz, A. Sahai, and B. Waters, "Predicate encryption supporting disjunctions, polynomial equations, and inner products," in *Advances in Cryptology—EUROCRYPT*. Berlin, Germany: Springer-Verlag, 2008, pp. 146–162.
[16] E. Shen, E. Shi, and B. Waters, "Predicate privacy in encryption systems," in *Proc. 6th Theory Cryptogr. Conf.* San Francisco, CA, USA: Springer-Verlag, 2009, pp. 457–473.
[17] A. Lewko, T. Okamoto, A. Sahai, K. Takashima, and B. Waters, "Fully secure functional encryption: Attribute-based encryption and (hierarchical) inner product encryption," in *Proc. 29th Annu. Int. Conf. Theory Appl. Cryptograph. Techn.* Edinburgh, U.K.: Springer-Verlag, 2010, pp. 62–91.
[18] A. Swaminathan *et al.*, "Confidentiality-preserving rank-ordered search," in *Proc. ACM Workshop Storage Secur. Survivability*. New York, NY, USA: ACM, 2007, pp. 7–12.
[19] C. Wang, N. Cao, K. Ren, and W. Lou, "Enabling secure and efficient ranked keyword search over outsourced cloud data," *IEEE Trans. Parallel Distrib. Syst.*, vol. 23, no. 8, pp. 1467–1479, Aug. 2012.
[20] N. Cao, C. Wang, M. Li, K. Ren, and W. Lou, "Privacy-preserving multi-keyword ranked search over encrypted cloud data," in *Proc. IEEE INFOCOM*, Apr. 2011, pp. 829–837.
[21] W. Sun *et al.*, "Privacy-preserving multi-keyword text search in the cloud supporting similarity-based ranking," in *Proc. 8th ACM SIGSAC Symp. Inf., Comput. Commun. Secur.* New York, NY, USA: ACM, 2013, pp. 71–82.
[22] Z. Fu, X. Sun, N. Linge, and L. Zhou, "Achieving effective cloud search services: Multi-keyword ranked search over encrypted cloud data supporting synonym query," *IEEE Trans. Consum. Electron.*, vol. 60, no. 1, pp. 164–172, Feb. 2014.
[23] X. Sun, X. Wang, Z. Xia, Z. Fu, and T. Li, "Dynamic multi-keyword top-k ranked search over encrypted cloud data," *J. Int. J. Secur. Appl.*, vol. 8, no. 1, pp. 319–332, 2014.
[24] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Commun. ACM*, vol. 13, no. 7, pp. 422–426, Jul. 1970.
[25] J. Bringer, H. Chabanne, and B. Kindarji, "Error-tolerant searchable encryption," in *Proc. IEEE Int. Conf. Commun.* Piscataway, NJ, USA: IEEE Press, Jun. 2009, pp. 1–6.
[26] M. Kuzu, M. S. Islam, and M. Kantarcioglu, "Efficient similarity search over encrypted data," in *Proc. IEEE 28th Int. Conf. Data Eng.*, Apr. 2012, pp. 1156–1167.
[27] H. T. Poon and A. Miri, "A low storage phase search scheme based on bloom filters for encrypted cloud services," in *Proc. IEEE 2nd Int. Conf. Cyber Secur. Cloud Comput.*, Nov. 2015, pp. 253–259.
[28] S. Zerr, E. Demidova, D. Olmedilla, W. Nejdl, M. Winslett, and S. Mitra, "Zerber: R-confidential indexing for distributed documents," in *Proc. 11th Int. Conf. Extending Database Technol., Adv. Database Technol.* New York, NY, USA: ACM, 2008, pp. 287–298.
[29] I. H. Witten, A. Moffat, and T. C. Bell, *Managing Gigabytes: Compressing and Indexing Documents and Images*. San Francisco, CA, USA: Morgan Kaufmann Publishing, 1999, pp. 36–56.
[30] D. A. Grossman, and O. Frieder, *Information Retrieval: Algorithms and Heuristics*, 2nd ed. Berlin, Germany: Springer, 2004, pp. 18–20.
[31] (2013). *NSF Research Awards Abstracts 1990–2003*. [Online]. Available: http://kdd.ics.uci.edu/databases/nsfabs/nsfawards.html

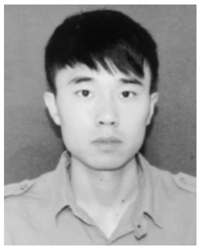


CHENG GUO received the B.S. degree in computer science from the Xi'an University of Architecture and Technology, in 2002, and the M.S. degree and the Ph.D. degree in computer application and technology from the Dalian University of Technology, Dalian, China, in 2006 and 2009, respectively, where he has been an Associate Professor with the School of Software Technology, since 2013. From 2010 to 2012, he held a postdoctoral position with the Department of Computer Science, National Tsing Hua University, Hsinchu, Taiwan. His current research interests include information security, cryptology, and cloud security.



RUHAN ZHUANG was born in Benxi, Liaoning, China, in 1994. She received the B.S. degree in software engineering from the Dalian University of Technology, China, in 2016, where she is currently pursuing the M.S. degree in software engineering. Her research interests include cryptography, private data protection technology, and cloud storage technology.

CHIN-CHEN CHANG received the B.S. degree in applied mathematics and the M.S. degree in computer and decision sciences from National Tsing Hua University, Hsinchu, Taiwan, in 1977 and 1979, respectively, and the Ph.D. degree in computer engineering from National Chiao Tung University, Hsinchu, in 1982. From 1983 to 1989, he was on the Faculty of the Institute of Applied Mathematics, National Chung Hsing University, Taichung, Taiwan. From 1989 to 1992, he was the Head and a Professor with the Institute of Computer Science and Information Engineering, National Chung Cheng University, Chiayi, Taiwan. From 1992 to 1995, he was the Dean of the College of Engineering, National Chung Cheng University. From 1995 to 1997, he was a Provost at National Chung Cheng University, where he was the Acting President, from 1996 to 1997. From 1998 to 2000, he was the Director of the Advisory Office of the Ministry of Education of the China. From 2002 to 2005, he was a Chair Professor with National Chung Cheng University. Since 2005, he has been a Chair Professor with Feng Chia University. He has served as a consultant to several research institutes and government departments. His current research interests include database design, computer cryptography, image compression, and data structures.



QIONGQIONG YUAN received the B.S. degree in software engineering from the Dalian University of Technology, Dalian, China, in 2015, where he is currently pursuing the M.S. degree with the School of Software Technology. His research interests include information security, cloud security, and cryptology.