IEEE Transactions on Parallel and Distributed Systems,Volume:PP,Issue:99,Date of Publication : 11.February.2015

A Secure and Dynamic Multi-keyword Ranked Search Scheme over Encrypted Cloud Data

Zhihua Xia, *Member, IEEE,* Xinhui Wang, Xingming Sun, *Senior Member, IEEE,* and Qian Wang, *Member, IEEE*

Abstract—Due to the increasing popularity of cloud computing, more and more data owners are motivated to outsource their data to cloud servers for great convenience and reduced cost in data management. However, sensitive data should be encrypted before outsourcing for privacy requirements, which obsoletes data utilization like keyword-based document retrieval. In this paper, we present a secure multi-keyword ranked search scheme over encrypted cloud data, which simultaneously supports dynamic update operations like deletion and insertion of documents. Specifically, the vector space model and the widely-used TF$\times$IDF model are combined in the index construction and query generation. We construct a special tree-based index structure and propose a "Greedy Depth-first Search" algorithm to provide efficient multi-keyword ranked search. The secure kNN algorithm is utilized to encrypt the index and query vectors, and meanwhile ensure accurate relevance score calculation between encrypted index and query vectors. In order to resist statistical attacks, phantom terms are added to the index vector for blinding search results . Due to the use of our special tree-based index structure, the proposed scheme can achieve sub-linear search time and deal with the deletion and insertion of documents flexibly. Extensive experiments are conducted to demonstrate the efficiency of the proposed scheme.

Index Terms—Searchable encryption, multi-keyword ranked search, dynamic update, cloud computing.

1 Introduction

Cloud computing has been considered as a new model of enterprise IT infrastructure, which can organize huge resource of computing, storage and applications, and enable users to enjoy ubiquitous, convenient and on-demand network access to a shared pool of configurable computing resources with great efficiency and minimal economic overhead [1]. Attracted by these appealing features, both individuals and enterprises are motivated to outsource their data to the cloud, instead of purchasing software and hardware to manage the data themselves.

Despite of the various advantages of cloud services, outsourcing sensitive information (such as e-mails, personal health records, company finance data, government documents, etc.) to remote servers brings privacy concerns. The cloud service providers (CSPs) that keep the data for users may access users' sensitive information without authorization. A general approach to protect the data confidentiality is to encrypt the data before outsourcing [2]. However, this will cause a huge cost in terms of data usability. For example, the existing techniques on keyword-based information retrieval, which are widely used on the plaintext data, cannot be directly applied on the encrypted data. Downloading all the data from the cloud and decrypt locally is obviously impractical.

In order to address the above problem, researchers have designed some general-purpose solutions with fully-homomorphic encryption [3] or oblivious RAMs [4]. However, these methods are not practical due to their high computational overhead for both the cloud sever and user. On the contrary, more practical special-purpose solutions, such as searchable encryption (SE) schemes have made specific contributions in terms of efficiency, functionality and security. Searchable encryption schemes enable the client to store the encrypted data to the cloud and execute keyword search over ciphertext domain. So far, abundant works have been proposed under different threat models to achieve various search functionality, such as single keyword search, similarity search, multi-keyword boolean search, ranked search, multi-keyword ranked search, etc. Among them, multi-keyword ranked search achieves more and more attention for its practical applicability. Recently, some *dynamic* schemes have been proposed to support inserting and deleting operations on document collection. These are significant works as it is highly possible that the data owners need to update their data on the cloud server. But few of the dynamic schemes support efficient multi-keyword ranked search.

This paper proposes a secure tree-based search scheme over the encrypted cloud data, which supports multi-keyword ranked search and dynamic operation on the document collection. Specifically, the vector space model and the widely-used "term frequency (TF) $\times$ inverse doc-

- *Zhihua Xia, Xinhui Wang, and Xingming Sun are with the Jiangsu Engineering Center of Network Monitoring, Jiangsu Collaborative Innovation Center on Atmospheric Environment and Equipment Technology, and School of Computer and Software, Nanjing University of Information Science & Technology, Nanjing, China.*
E-mail: xia_zhihua, wxh_nuist, sunnudt@163.com.
- *Qian Wang is with the the School of Computer, Wuhan University, Wuhan, China.*
E-mail: qianwang@whu.edu.cn.

ument frequency (IDF)" model are combined in the index construction and query generation to provide multi-keyword ranked search. In order to obtain high search efficiency, we construct a tree-based index structure and propose a "Greedy Depth-first Search" algorithm based on this index tree. Due to the special structure of our tree-based index, the proposed search scheme can flexibly achieve sub-linear search time and deal with the deletion and insertion of documents. The secure kNN algorithm is utilized to encrypt the index and query vectors, and meanwhile ensure accurate relevance score calculation between encrypted index and query vectors. To resist different attacks in different threat models, we construct two secure search schemes: the basic dynamic multi-keyword ranked search (BDMRS) scheme in the known ciphertext model, and the enhanced dynamic multi-keyword ranked search (EDMRS) scheme in the known background model. Our contributions are summarized as follows:

1) We design a searchable encryption scheme that supports both the accurate multi-keyword ranked search and flexible dynamic operation on document collection.
2) Due to the special structure of our tree-based index, the search complexity of the proposed scheme is fundamentally kept to logarithmic. And in practice, the proposed scheme can achieve higher search efficiency by executing our "Greedy Depth-first Search" algorithm. Moreover, parallel search can be flexibly performed to further reduce the time cost of search process.

The reminder of this paper is organized as follows. Related work is discussed in Section 2, and Section 3 gives a brief introduction to the system model, threat model, the design goals, and the preliminaries. Section 4 describes the schemes in detail. Section 5 presents the experiments and performance analysis. And Section 6 covers the conclusion.

2 Related Work

Searchable encryption schemes enable the clients to store the encrypted data to the cloud and execute keyword search over ciphertext domain. Due to different cryptography primitives, searchable encryption schemes can be constructed using public key based cryptography [5], [6] or symmetric key based cryptography [7], [8], [9], [10].

Song *et al.* [7] proposed the first symmetric searchable encryption (SSE) scheme, and the search time of their scheme is linear to the size of the data collection. Goh [8] proposed formal security definitions for SSE and designed a scheme based on Bloom filter. The search time of Goh's scheme is $O(n)$, where n is the cardinality of the document collection. Curtmola *et al.* [10] proposed two schemes (SSE-1 and SSE-2) which achieve the optimal search time. Their SSE-1 scheme is secure against chosen-keyword attacks (CKA1) and SSE-2 is secure against adaptive chosen-keyword attacks (CKA2). These early works are single keyword boolean search schemes, which are very simple in terms of functionality. Afterward, abundant works have been proposed under different threat models to achieve various search functionality, such as single keyword search, similarity search [11], [12], [13], [14], multi-keyword boolean search [15], [16], [17], [18], [19], [20], [21], [22], ranked search [23], [24], [25], and multi-keyword ranked search [26], [27], [28], [29], etc.

Multi-keyword boolean search allows the users to input multiple query keywords to request suitable documents. Among these works, conjunctive keyword search schemes [15], [16], [17] only return the documents that contain all of the query keywords. Disjunctive keyword search schemes [18], [19] return all of the documents that contain a subset of the query keywords. Predicate search schemes [20], [21], [22] are proposed to support both conjunctive and disjunctive search. All these multi-keyword search schemes retrieve search results based on the existence of keywords, which cannot provide acceptable result ranking functionality.

Ranked search can enable quick search of the most relevant data. Sending back only the top-k most relevant documents can effectively decrease network traffic. Some early works [23], [24], [25] have realized the ranked search using order-preserving techniques, but they are designed only for single keyword search. Cao *et al.* [26] realized the first privacy-preserving multi-keyword ranked search scheme, in which documents and queries are represented as vectors of dictionary size. With the "coordinate matching", the documents are ranked according to the number of matched query keywords. However, Cao *et al.*'s scheme does not consider the importance of the different keywords, and thus is not accurate enough. In addition, the search efficiency of the scheme is linear with the cardinality of document collection. Sun *et al.* [27] presented a secure multi-keyword search scheme that supports similarity-based ranking. The authors constructed a searchable index tree based on vector space model and adopted cosine measure together with TF$\times$IDF to provide ranking results. Sun *et al.*'s search algorithm achieves better-than-linear search efficiency but results in precision loss. Örencik *et al.* [28] proposed a secure multi-keyword search method which utilized local sensitive hash (LSH) functions to cluster the similar documents. The LSH algorithm is suitable for similar search but cannot provide exact ranking. In [29], Zhang *et al.* proposed a scheme to deal with secure multi-keyword ranked search in a multi-owner model. In this scheme, different data owners use different secret keys to encrypt their documents and keywords while authorized data users can query without knowing keys of these different data owners. The authors proposed an "Additive Order Preserving Function" to retrieve the most relevant search results. However, these works don't support dynamic operations.

Practically, the data owner may need to update the document collection after he upload the collection to the cloud server. Thus, the SE schemes are expected to support the insertion and deletion of the documents. There are also several dynamic searchable encryption schemes. In the work of Song *et al.* [7], the each document is considered as a sequence of fixed length words, and is individually indexed. This scheme supports straightforward update operations but with low efficiency. Goh [8] proposed a scheme to generate a sub-index (Bloom filter) for every document based on keywords. Then the dynamic operations can be easily realized through updating of a Bloom filter along with the corresponding document. However, Goh's scheme has linear search time and suffers from false positives. In 2012, Kamara *et al.* [30] constructed an encrypted inverted index that can handle dynamic data efficiently. But, this scheme is very complex to implement. Subsequently, as an improvement, Kamara *et al.* [31] proposed a new search scheme based on tree-based index, which can handle dynamic update on document data stored in leaf nodes. However, their scheme is designed only for single-keyword Boolean search. In [32], Cash *et al.* presented a data structure for keyword/identity tuple named "T-Set". Then, a document can be represented by a series of independent T-Sets. Based on this structure, Cash *et al.* [33] proposed a dynamic searchable encryption scheme. In their construction, newly added tuples are stored in another database in the cloud, and deleted tuples are recorded in a revocation list. The final search result is achieved through excluding tuples in the revocation list from the ones retrieved from original and newly added tuples. Yet, Cash *et al.*'s dynamic search scheme doesn't realize the multi-keyword ranked search functionality.

3 Problem Formulation

3.1 Notations and Preliminaries

- $\mathcal{W}$ – The dictionary, namely, the set of keywords, denoted as $\mathcal{W} = \{w_1, w_2, ..., w_m\}$.
- m – The total number of keywords in $\mathcal{W}$.
- $\mathcal{W}_q$ – The subset of $\mathcal{W}$, representing the keywords in the query.
- $\mathcal{F}$ – The plaintext document collection, denoted as a collection of n documents $\mathcal{F} = \{f_1, f_2, ..., f_n\}$. Each document f in the collection can be considered as a sequence of keywords.
- n – The total number of documents in $\mathcal{F}$.
- $\mathcal{C}$ – The encrypted document collection stored in the cloud server, denoted as $\mathcal{C} = \{c_1, c_2, ..., c_n\}$.
- $\mathcal{T}$ – The unencrypted form of index tree for the whole document collection $\mathcal{F}$.
- $\mathcal{I}$ – The searchable encrypted tree index generated from $\mathcal{T}$.
- Q – The query vector for keyword set $\mathcal{W}_q$.
- TD – The encrypted form of Q, which is named as trapdoor for the search request.
- D_u – The index vector stored in tree node u whose dimension equals to the cardinality of the dictionary $\mathcal{W}$. Note that the node u can be either a leaf node or an internal node of the tree.
- I_u – The encrypted form of D_u.

Vector Space Model and Relevance Score Function. Vector space model along with TF×IDF rule is widely used in plaintext information retrieval, which efficiently supports ranked multi-keyword search [34]. Here, the term frequency (TF) is the number of times a given term (keyword) appears within a document, and the inverse document frequency (IDF) is obtained through dividing the cardinality of document collection by the number of documents containing the keyword. In the vector space model, each document is denoted by a vector, whose elements are the normalized TF values of keywords in this document. Each query is also denoted as a vector Q, whose elements are the normalized IDF values of query keywords in the document collection. Naturally, the lengths of both the TF vector and the IDF vector are equal to the total number of keywords, and the dot product of the TF vector D_u and the IDF vector Q can be calculated to quantify the relevance between the query and corresponding document. Following are the notations used in our relevance evaluation function:

- N_{f,w_i} – The number of keyword w_i in document f.
- N – The total number of documents.
- N_{w_i} – The number of documents that contain keyword w_i.
- TF'_{f,w_i} – The TF value of w_i in document f.
- IDF'_{w_i} – The IDF value of w_i in document collection.
- TF_{u,w_i} – The normalized TF value of keyword w_i stored in index vector D_u.
- IDF_{w_i} – The normalized IDF value of keyword w_i in document collection.

The relevance evaluation function is defined as:

$$\text{RScore}(D_u, Q) = D_u \cdot Q = \sum_{w_i \in \mathcal{W}_q} TF_{u,w_i} \times IDF_{w_i}. \quad (1)$$

If u is an internal node of the tree, TF_{u,w_i} is calculated from index vectors in the child nodes of u. If the u is a leaf node, TF_{u,w_i} is calculated as:

$$TF_{u,w_i} = \frac{TF'_{f,w_i}}{\sqrt{\sum_{w_i \in \mathcal{W}} (TF'_{f,w_i})^2}}, \quad (2)$$

where $TF'_{f,w_i} = 1 + \ln N_{f,w_i}$. And in the search vector Q, IDF_{w_i} is calculated as:

$$IDF_{w_i} = \frac{IDF'_{w_i}}{\sqrt{\sum_{w_i \in \mathcal{W}_q} (IDF'_{w_i})^2}}, \quad (3)$$

where $IDF'_{w_i} = \ln(1 + N/N_{w_i})$.

Keyword Balanced Binary Tree. The balanced binary tree is widely used to deal with optimization problems [35], [36]. The keyword balanced binary (KBB) tree in our scheme is a dynamic data structure whose node stores a vector D. The elements of vector D are the normalized TF values. Sometimes, we refer the vector D in the node

u to D_u for simplicity. Formally, the node u in our KBB tree is defined as follows:

$$u = \langle ID, D, P_l, P_r, FID \rangle, \tag{4}$$

where ID denotes the identity of node u, P_l and P_r are respectively the pointers to the left and right child of node u. If the node u is a leaf node of the tree, FID stores the identity of a document, and D denotes a vector consisting of the normalized TF values of the keywords to the document. If the node u is an internal node, FID is set to $null$, and D denotes a vector consisting of the TF values which is calculated as follows:

$$D[i] = max\{u.P_l \to D[i], u.P_r \to D[i]\}, i = 1, ..., m. \tag{5}$$

The detailed construction process of the tree-based index is illustrated in Section 4, which is denoted as BuildIndexTree($\mathcal{F}$).

3.2 The System and Threat Models

The system model in this paper involves three different entities: data owner, data user and cloud server, as illustrated in Fig. 1.

Data owner has a collection of documents $\mathcal{F} = \{f_1, f_2, ..., f_n\}$ that he wants to outsource to the cloud server in encrypted form while still keeping the capability to search on them for effective utilization. In our scheme, the data owner firstly builds a secure searchable tree index $\mathcal{I}$ from document collection $\mathcal{F}$, and then generates an encrypted document collection $\mathcal{C}$ for $\mathcal{F}$. Afterwards, the data owner outsources the encrypted collection $\mathcal{C}$ and the secure index $\mathcal{I}$ to the cloud server, and securely distributes the key information of trapdoor generation (including keyword IDF values) and document decryption to the authorized data users.

Besides, the data owner is responsible for the update operation of his documents stored in the cloud server. While updating, the data owner generates the update information locally and sends it to the server.

Data users are authorized ones to access the documents of data owner. With t query keywords, the authorized user can generate a trapdoor TD according to search control mechanisms to fetch k encrypted documents from cloud server. Then, the data user can decrypt the documents with the shared secret key.

Cloud server stores the encrypted document collection $\mathcal{C}$ and the encrypted searchable tree index $\mathcal{I}$ for data owner. Upon receiving the trapdoor TD from the data user, the cloud server executes search over the index tree $\mathcal{I}$, and finally returns the corresponding collection of top-k ranked encrypted documents. Besides, upon receiving the update information from the data owner, the server needs to update the index $\mathcal{I}$ and document collection $\mathcal{C}$ according to the received information.

The cloud server in the proposed scheme is considered as "honest-but-curious", which is employed by lots of works on secure cloud data search [25], [26], [27]. Specifically, the cloud server honestly and correctly executes instructions in the designated protocol. Meanwhile, it is curious to infer and analyze received data, which helps it acquire additional information. Depending on what information the cloud server knows, we adopt the two threat models proposed by Cao *et al.* [26].

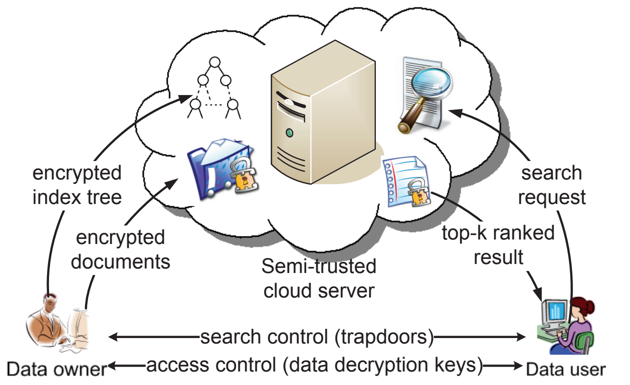


Fig. 1. The architecture of ranked search over encrypted cloud data

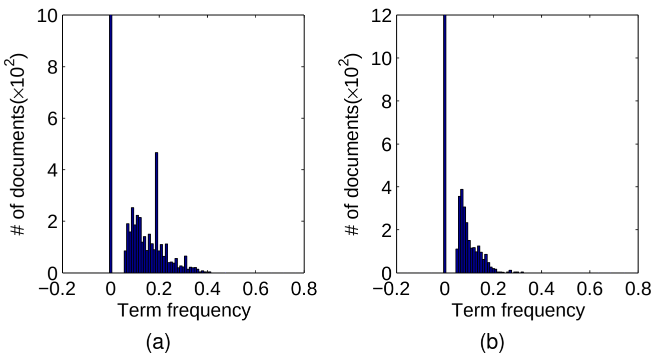


Fig. 2. Distribution of term frequency (TF) for (a) keyword "subnet", and (b) keyword "host".

Known Ciphertext Model. In this model, the cloud server only knows the encrypted document collection $\mathcal{C}$, the searchable index tree $\mathcal{I}$, and the search trapdoor TD submitted by the authorized user. That is to say, the cloud server can conduct ciphertext-only attack (COA) [37] in this model.

Known Background Model. Compared with known ciphertext model, the cloud server in this stronger model is equipped with more knowledge, such as the term frequency (TF) statistics of the document collection. This statistical information records how many documents are there for each term frequency of a specific keyword in the whole document collection, as shown in Fig. 2, which could be used as the keyword identity. Equipped with such statistical information, the cloud server can conduct TF statistical attack to deduce or even identify certain keywords through analyzing histogram and value range of the corresponding frequency distributions [24], [25], [27].

3.3 Design Goals

To enable secure, efficient, accurate and dynamic multi-keyword ranked search over outsourced encrypted cloud

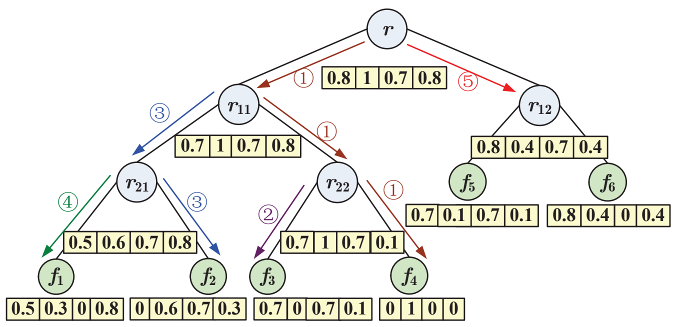


Fig. 3. An example of the tree-based index with the document collection $\mathcal{F} = \{f_i|i = 1, ..., 6\}$ and cardinality of the dictionary $m = 4$. In the construction process of the tree index, we first generate leaf nodes from the documents. Then, the internal tree nodes are generated based on the leaf nodes. This figure also shows an example of search process, in which the query vector Q is equal to $(0, 0.92, 0, 0.38)$. In this example, we set the parameter $k = 3$ with the meaning that three documents will be returned to the user. According to the search algorithm, the search starts with the root node, and reaches the first leaf node f_4 through r_{11} and r_{22}. The relevance score of f_4 to the query is 0.92. After that, the leaf nodes f_3 and f_2 are successively reached with the relevance scores 0.038 and 0.67. Next, the leaf node f_1 is reached with score 0.58 and replace f_3 in $RList$. Finally, the algorithm will try to search subtree rooted by r_{12}, and find that there are no reasonable results in this subtree because the relevance score of r_{12} is 0.52, which is smaller than the smallest relevance score in $RList$.

data under the above models, our system has the following design goals.

Dynamic: The proposed scheme is designed to provide not only multi-keyword query and accurate result ranking, but also dynamic update on document collections.

Search Efficiency: The scheme aims to achieve sublinear search efficiency by exploring a special tree-based index and an efficient search algorithm.

Privacy-preserving: The scheme is designed to prevent the cloud server from learning additional information about the document collection, the index tree, and the query. The specific privacy requirements are summarized as follows,

1) *Index Confidentiality and Query Confidentiality:* The underlying plaintext information, including keywords in the index and query, TF values of keywords stored in the index, and IDF values of query keywords, should be protected from cloud server;
2) *Trapdoor Unlinkability:* The cloud server should not be able to determine whether two encrypted queries (trapdoors) are generated from the same search request;
3) *Keyword Privacy:* The cloud server could not identify the specific keyword in query, index or document collection by analyzing the statistical information like term frequency. Note that our proposed scheme is not designed to protect access pattern, i.e., the sequence of returned documents.

4 The Proposed Schemes

In this section, we firstly describe the unencrypted dynamic multi-keyword ranked search (UDMRS) scheme which is constructed on the basis of vector space model and KBB tree. Based on the UDMRS scheme, two secure search schemes (BDMRS and EDMRS schemes) are constructed against two threat models, respectively.

4.1 Index Construction of UDMRS Scheme

In Section 3, we have briefly introduced the KBB index tree structure, which assists us in introducing the index construction. In the process of index construction, we first generate a tree node for each document in the collection. These nodes are the leaf nodes of the index tree. Then, the internal tree nodes are generated based on these leaf nodes. The formal construction process of the index is presented in Algorithm 1. An example of our index tree is shown in Fig. 3. Note that the index tree $\mathcal{T}$ built here is a plaintext.

Following are some notations for Algorithm 1. Besides, the data structure of the tree node is defined as $\langle ID, D, P_l, P_r, FID\rangle$, where the unique identity ID for each tree node is generated through the function GenID().

- $CurrentNodeSet$ – The set of current processing nodes which have no parents. If the number of nodes is even, the cardinality of the set is denoted as $2h (h \in \mathbf{Z}^+)$, else the cardinality is denoted as $(2h + 1)$.
- $TempNodeSet$ – The set of the newly generated nodes.

In the index, if $D_u[i] \neq 0$ for an internal node u, there is at least one path from the node u to some leaf, which indicates a document containing the keyword w_i. In addition, $D_u[i]$ always stores the biggest normalized TF value of w_i among its child nodes. Thus, the possible largest relevance score of its children can be easily estimated.

4.2 Search Process of UDMRS Scheme

The search process of the UDMRS scheme is a recursive procedure upon the tree, named as "Greedy Depth-first Search (GDFS)" algorithm. We construct a result list denoted as $RList$, whose element is defined as $\langle RScore, FID\rangle$. Here, the $RScore$ is the relevance score of the document f_{FID} to the query, which is calculated according to Formula (1). The $RList$ stores the k accessed documents with the largest relevance scores to the query. The elements of the list are ranked in descending order according to the $RScore$, and will be updated timely during the search process. Following are some other notations, and the GDFS algorithm is described in Algorithm 2.

- RScore(D_u, Q) – The function to calculate the relevance score for query vector Q and index vector D_u stored in node u, which is defined in Formula (1).
- $k^{th}score$ – The smallest relevance score in current $RList$, which is initialized as 0.
- $hchild$ – The child node of a tree node with higher relevance score.

Algorithm 1 BuildIndexTree($\mathcal{F}$)

```
Input: the document collection $\mathcal{F} = \{f_1, f_2, ..., f_n\}$ with the identifiers $\mathcal{FID} = \{FID|FID = 1, 2, ..., n\}$.
Output: the index tree $\mathcal{T}$
1:  for each document $f_{FID}$ in $\mathcal{F}$ do
2:      Construct a leaf node $u$ for $f_{FID}$, with $u.ID$ = GenID(), $u.P_l = u.P_r = null$, $u.FID = FID$, and $D[i] = TF_{f_{FID},w_i}$ for $i = 1, ..., m$;
3:      Insert $u$ to $CurrentNodeSet$;
4:  end for
5:  while the number of nodes in $CurrentNodeSet$ is larger than 1 do
6:      if the number of nodes in $CurrentNodeSet$ is even, i.e. $2h$ then
7:          for each pair of nodes $u'$ and $u''$ in $CurrentNodeSet$ do
8:              Generate a parent node $u$ for $u'$ and $u''$, with $u.ID$ = GenID(), $u.P_l = u'$, $u.P_r = u''$, $u.FID = 0$ and $D[i] = max\{u'.D[i], u''.D[i]\}$ for each $i = 1, ..., m$;
9:              Insert $u$ to $TempNodeSet$;
10:         end for
11:     else
12:         for each pair of nodes $u'$ and $u''$ of the former $(2h-2)$ nodes in $CurrentNodeSet$ do
13:             Generate a parent node $u$ for $u'$ and $u''$;
14:             Insert $u$ to $TempNodeSet$;
15:         end for
16:         Create a parent node $u_1$ for the $(2h-1)$-th and $2h$-th node, and then create a parent node $u$ for $u_1$ and the $(2h+1)$-th node;
17:         Insert $u$ to $TempNodeSet$;
18:     end if
19:     Replace $CurrentNodeSet$ with $TempNodeSet$ and then clear $TempNodeSet$;
20: end while
21: return the only node left in $CurrentNodeSet$, namely, the root of index tree $\mathcal{T}$;
```

Algorithm 2 GDFS(IndexTreeNode u)

```
1:  if the node $u$ is not a leaf node then
2:      if RScore$(D_u, Q) > k^{th}score$ then
3:          GDFS$(u.hchild)$;
4:          GDFS$(u.lchild)$;
5:      else
6:          return
7:      end if
8:  else
9:      if RScore$(D_u, Q) > k^{th}score$ then
10:         Delete the element with the smallest relevance score from $RList$;
11:         Insert a new element $\langle$RScore$(D_u, Q), u.FID\rangle$ and sort all the elements of $RList$;
12:     end if
13:     return
14: end if
```

- $lchild$ – The child node of a tree node with lower relevance score.

Since the possible largest relevance score of documents rooted by the node u can be predicted, only a part of the nodes in the tree are accessed during the search process. Fig. 3 shows an example of search process with the document collection $\mathcal{F} = \{f_i | i = 1, ..., 6\}$, cardinality of the dictionary $m = 4$, and query vector $Q = (0, 0.92, 0, 0.38)$.

4.3 BDMRS Scheme

Based on the UDMRS scheme, we construct the basic dynamic multi-keyword ranked search (BDMRS) scheme by using the secure kNN algorithm [38]. The BDMRS scheme is designed to achieve the goal of privacy-preserving in the known ciphertext model, and the four algorithms included are described as follows:

- $\mathcal{SK} \leftarrow$ Setup() Initially, the data owner generates the secret key set $\mathcal{SK}$, including 1) a randomly generated m-bit vector S where m is equal to the cardinality of dictionary, and 2) two $(m \times m)$ invertible matrices M_1 and M_2. Namely, $\mathcal{SK} = \{S, M_1, M_2\}$.
- $\mathcal{I} \leftarrow$ GenIndex$(\mathcal{F}, \mathcal{SK})$ First, the unencrypted index tree $\mathcal{T}$ is built on $\mathcal{F}$ by using $\mathcal{T} \leftarrow$ BuildIndexTree$(\mathcal{F})$. Secondly, the data owner generates two random vectors $\{D_u', D_u''\}$ for index vector D_u in each node u, according to the secret vector S. Specifically, if $S[i] = 0$, $D_u'[i]$ and $D_u''[i]$ will be set equal to $D_u[i]$; if $S[i] = 1$, $D_u'[i]$ and $D_u''[i]$ will be set as two random values whose sum equals to $D_u[i]$. Finally, the encrypted index tree $\mathcal{I}$ is built where the node u stores two encrypted index vectors $I_u = \{M_1^T D_u', M_2^T D_u''\}$.
- $TD \leftarrow$ GenTrapdoor$(\mathcal{W}_q, \mathcal{SK})$ With keyword set $\mathcal{W}_q$, the unencrypted query vector Q with length of m is generated. If $w_i \in \mathcal{W}_q$, $Q[i]$ stores the normalized IDF value of w_i; else $Q[i]$ is set to 0. Similarly, the query vector Q is split into two random vectors Q' and Q''. The difference is that if $S[i] = 0$, $Q'[i]$ and $Q''[i]$ are set to two random values whose sum equals to $Q[i]$; else $Q'[i]$ and $Q''[i]$ are set as the same as $Q[i]$. Finally, the algorithm returns the trapdoor $TD = \{M_1^{-1}Q', M_2^{-1}Q''\}$.
- $RelevanceScore \leftarrow$ SRScore(I_u, TD) With the trapdoor TD, the cloud server computes the relevance score of node u in the index tree $\mathcal{I}$ to the query. Note that the relevance score calculated from encrypted vectors is equal to that from unencrypted vectors as follows:

$$\begin{aligned}
& I_u \cdot TD \\
&= (M_1^T D_u') \cdot (M_1^{-1}Q') + (M_2^T D_u'') \cdot (M_2^{-1}Q'') \\
&= (M_1^T D_u')^T (M_1^{-1}Q') + (M_2^T D_u'')^T (M_2^{-1}Q'') \\
&= D_u'^T M_1 M_1^{-1} Q' + D_u''^T M_2 M_2^{-1} Q'' \\
&= D_u' \cdot Q' + D_u'' \cdot Q'' \\
&= D_u \cdot Q \\
&= \text{RScore}(D_u, Q)
\end{aligned} \tag{6}$$

Security analysis. We analyze the BDMRS scheme according to the three predefined privacy requirements in the design goals:

1) *Index Confidentiality and Query Confidentiality:* In the proposed BDMRS scheme, I_u and TD are obfuscated vectors, which means the cloud server cannot infer the original vectors D_u and Q without the secret key set $\mathcal{SK}$. The secret keys M_1 and M_2 are Gaussian random matrices. According to [38], the attacker (cloud server) of COA cannot calculate the matrices merely with ciphertext. Thus, the BDMRS scheme is resilient against ciphertext-only attack (COA) and the index confidentiality and the query confidentiality are well protected.
2) *Query Unlinkability:* The trapdoor of query vector is generated from a random splitting operation, which means that the same search requests will be transformed into different query trapdoors, and thus the query unlinkability is protected. However, the cloud server is able to link the same search requests according to the same visited path and the same relevance scores.
3) *Keyword Privacy:* In this scheme, the confidentiality of the index and query are well protected that the original vectors are kept from the cloud server. And the search process merely introduces inner product computing of encrypted vectors, which leaks no information about any specific keyword. Thus, the keyword privacy is protected in the known ciphertext model. But in the known background model, the cloud server is supposed to have more knowledge, such as the term frequency statistics of keywords. This statistic information can be visualized as a TF distribution histogram which reveals how many documents are there for every TF value of a specific keyword in the document collection. Then, due to the specificity of the TF distribution histogram, like the graph slope and value range, the cloud server could conduct TF statistical attack to deduce/identify keywords [25], [24], [27]. In the worst case, when there is only one keyword in the query vector, i.e. the normalized IDF value for the keyword is 1, the final relevance score distribution is exactly the normalized TF distribution of this keyword, which is directly exposed to cloud server. Therefore, the BDMRS scheme cannot resist TF statistical attack in the known background model.

4.4 EDMRS Scheme

The security analysis above shows that the BDMRS scheme can protect the *Index Confidentiality and Query Confidentiality* in the known ciphertext model. However, the cloud server is able to link the same search requests by tracking path of visited nodes. In addition, in the known background model, it is possible for the cloud server to identify a keyword as the normalized TF distribution of the keyword can be exactly obtained from the final calculated relevance scores. The primary cause is that the relevance score calculated from I_u and TD is exactly equal to that from D_u and Q. A heuristic method to further improve the security is to break such exact equality. Thus, we can introduce some tunable randomness to disturb the relevance score calculation. In addition, to suit different users' preferences for higher accurate ranked results or better protected keyword privacy, the randomness are set adjustable.

The enhanced EDMRS scheme is almost the same as BDMRS scheme except that:

- $\mathcal{SK} \leftarrow \mathsf{Setup}()$ In this algorithm, we set the secret vector S as a m-bit vector, and set M_1 and M_2 are $(m + m') \times (m + m')$ invertible matrices, where m' is the number of phantom terms.
- $\mathcal{I} \leftarrow \mathsf{GenIndex}(\mathcal{F}, \mathcal{SK})$ Before encrypting the index vector D_u, we extend the vector D_u to be a $(m+m')$-dimensional vector. Each extended element $D_u[m + j]$, $j = 1, ..., m'$, is set as a random number ε_j.
- $TD \leftarrow \mathsf{GenTrapdoor}(\mathcal{W}_q, \mathcal{SK})$ The query vector Q is extended to be a $(m + m')$-dimensional vector. Among the extended elements, a number of m'' elements are randomly chosen to set as 1, and the rest are set as 0.
- $RelevanceScore \leftarrow \mathsf{SRScore}(I_u, TD)$ After the execution of relevance evaluation by cloud server, the final relevance score for index vector I_u equals to $D_u \cdot Q + \sum \varepsilon_v$, where $v \in \{j|Q[m + j] = 1\}$.

Security analysis. The security of EDMRS scheme is also analyzed according to the three predefined privacy requirements in the design goals:

1) *Index Confidentiality and Query Confidentiality:* Inherited from BDMRS scheme, the EDMRS scheme can protect index confidentiality and query confidentiality in the known background model. Due to the utilization of phantom terms, the confidentiality is further enhanced as the transformation matrices are harder to figure out [38].
2) *Query Unlinkability:* By introducing the random value ε, the same search requests will generate different query vectors and receive different relevance score distributions. Thus, the query unlinkability is protected better. However, since the proposed scheme is not designed to protect access pattern for efficiency issues, the motivated cloud server can analyze the similarity of search results to judge whether the retrieved results come from the same requests. In the proposed EDMRS scheme, the data user can control the level of unlinkability by adjusting the value of $\sum \varepsilon_v$. This is a trade-off between accuracy and privacy, which is determined by the user.
3) *Keyword Privacy:* As is discussed in Section 4.3, the BDMRS scheme cannot resist TF statistical attack in the known background model, as the cloud server is able to deduce/identify keywords through analyzing the TF distribution histogram. Thus, the

TABLE 1
The change of keyword IDF values after updating in a collection with 5000 documents.

Keyword NO	Original IDF values	IDF values in the updated collection			
		After deleting 100 documents	After deleting 300 documents	After adding 100 documents	After adding 300 documents
1	3.0332	3.0253	3.0166	3.0334	3.0267
2	3.2581	3.2581	3.2530	3.2628	3.2857
3	3.7616	3.7584	3.7431	3.7647	3.7550
4	3.8934	3.8926	3.8910	3.9128	3.9226
5	5.6304	5.6103	5.6861	5.6501	5.6885
6	5.7478	5.7277	5.6861	5.7675	5.8059
7	5.8121	5.7920	5.8192	5.8319	5.8702
8	7.4192	7.3990	7.3573	7.4390	7.4774
9	7.8244	7.8043	7.7626	7.8442	7.8827
10	8.5174	8.4972	8.4555	8.5372	8.5757

EDMRS scheme is designed to obscure the TF distributions of keywords with the randomness of $\sum \varepsilon_v$. In order to maximize the randomness of relevance score distributions, we need to get as many different $\sum \varepsilon_v$ as possible. Given that there are 2^ω different choices of $\sum \varepsilon_v$ for each index vector, the possibility that two $\sum \varepsilon_v$ sharing the same value is $1/2^\omega$. In the EDMRS scheme, the number of different $\sum \varepsilon_v$ is equal to $\binom{m'}{m''}$, which reaches the maximum when $m'/m'' = 2$. Hence, considering $\binom{m'}{m''} \geq (m'/m'')^{m''} = 2^{m''}$, we set $m' = 2\omega$ and $m'' = \omega$ so that the number of different $\sum \varepsilon_v$ is greater than 2^ω. Therefore, there are at least 2ω dummy elements in every vector, and half of them need to be randomly selected to generate $\sum \varepsilon_v$ in each query. In addition, we set every ε_j to follow the same uniform distribution $U(\mu' - \delta, \mu' + \delta)$. According to the central limit theorem, the $\sum \varepsilon_v$ follows the normal distribution $N(\mu, \sigma^2)$, where expectation μ and standard deviation σ can be calculated as:

$$\begin{cases} \mu = \omega\mu' \\ \sigma^2 = \omega\delta^2/3. \end{cases} \tag{7}$$

In the real application, we can set $\mu = 0$, and balance the accuracy and privacy by adjusting the variance σ.

4.5 Dynamic Update Operation of DMRS

After insertion or deletion of a document, we need to update synchronously the index. Since the index of DMRS scheme is designed as a balanced binary tree, the dynamic operation is carried out by updating nodes in the index tree. Note that the update on index is merely based on document identifies, and no access to the content of documents is required. The specific process is presented as follows:

- $\{\mathcal{I}'_s, c_i\} \leftarrow$ GenUpdateInfo$(\mathcal{SK}, \mathcal{T}_s, i, updtype))$ This algorithm generates the update information $\{\mathcal{I}'_s, c_i\}$ which will be sent to the cloud server. In order to reduce the communication overhead, the data owner stores a copy of unencrypted index tree. Here, the notion $updtype \in \{Ins, Del\}$ denotes either an insertion or a deletion for the document f_i. The notion $\mathcal{T}_s$ denotes the set consisting of the tree nodes that need to be changed during the update. For example, if we want to delete the document f_4 in Fig. 3, the subtree $\mathcal{T}_s$ includes a set of nodes $\{r_{22}, r_{11}, r\}$.
  - If $updtype$ is equal to Del, the data owner deletes from the subtree the leaf node that stores the document identity i and updates the vector D of other nodes in subtree $\mathcal{T}_s$, so as to generate the updated subtree $\mathcal{T}'_s$. In particular, if the deletion of the leaf node breaks the balance of the binary index tree, we replace the deleted node with a fake node whose vector is padded with 0 and file identity is $null$. Then, the data owner encrypts the vectors stored in the subtree $\mathcal{T}'_s$ with the key set $\mathcal{SK}$ to generate encrypted subtree $\mathcal{I}'_s$, and set the output c_i as $null$.
  - If $updtype$ is equal to Ins, the data owner generates a tree node $u = \langle \text{GenID}(), D, null, null, i\rangle$ for the document f_i, where $D[j] = TF_{f_i,w_j}$ for $j = 1, ..., m$. Then, the data owner inserts this new node into the subtree $\mathcal{T}_s$ as a leaf node and updates the vector D of other nodes in subtree $\mathcal{T}_s$ according to the Formula (5), so as to generate the new subtree $\mathcal{T}'_s$. Here, the data owner is always preferable to replace the fake leaf nodes generated by Del operation with newly inserted nodes, instead of directly inserting new nodes. Next, the data owner encrypts the vectors stored in subtree $\mathcal{T}'_s$ with the key set $\mathcal{SK}$ as described in Section 4.4, to generate encrypted subtree $\mathcal{I}'_s$. Finally, the document f_i is encrypted to c_i.
- $\{\mathcal{I}', \mathcal{C}'\} \leftarrow$ Update$(\mathcal{I}, \mathcal{C}, updtype, \mathcal{I}'_s, c_i)$ In this algorithm, cloud server replaces the corresponding subtree $\mathcal{I}_s$(the encrypted form of $\mathcal{T}_s$) with $\mathcal{I}'_s$, so as to generate a new index tree $\mathcal{I}'$. If $updtype$ is equal to Ins, cloud server inserts the encrypted document c_i into $\mathcal{C}$, obtaining a new collection $\mathcal{C}'$. If $updtype$ is equal to Del, cloud server deletes the encrypted document c_i from $\mathcal{C}$ to obtain the new collection $\mathcal{C}'$.

Similar to the scheme in [31], our scheme can also carry out the update operation without storing the index tree on data owner side. We choose to store the unencrypted index tree on the data owner side to tradeoff storage cost for less communication burdens. In both of the Kamara et al.'s scheme [31] and our design, it needs to change a set of nodes to update a leaf node because the vector data of an internal node is computed from its children. If the data owner does not store the unencrypted subtree, the whole update process needs two rounds of communications between the cloud server and the data owner. Specifically, the data owner should firstly download the involved subtree in encrypted form from the cloud server. Secondly, the data owner decrypts the subtree and updates it with the newly added or deleted leaf node. Thirdly, the data owner re-encrypts the subtree and uploads the encrypted subtree to the cloud server. Finally, the cloud server replaces the old subtree with the updated one. Thus, to reduce the communication cost, we store an unencrypted tree on the data owner side. Then, the data owner can update the subtree directly with the newly added or deleted leaf node and encrypt and upload the updated subtree to the cloud server. In this case, the update operation can be finished with one round of communication between the cloud server and the data owner.

As a dynamic scheme, it is not reasonable to fix the length of vector as the size of dictionary because the newly-added document may contain the keywords out of the dictionary. In the proposed scheme, we add some blank entries in the dictionary and set corresponding entries in each index vector as 0. If new keywords appear while inserting documents, these blank entries are replaced with new keywords. Then, the index vectors of newly added documents are generated based on the updated dictionary, while the other index vectors are not affected and remain the same as before.

After several times of document updating, the real IDF values of some keywords in the present collection may have obviously changed. Therefore, as the distributor of the IDF data, the data owner needs to recalculate the IDF values for all keywords and distribute them to authorized users. In Table 1, there are three classes of keywords with different IDF value ranges. The smaller IDF value means the keyword appears more frequently. Table 1 shows that after adding or deleting 100 and 300 documents, the IDF values do not change a lot. Thus, the data owner is unnecessary to update IDF values every time when he executes update operation on the dataset. The data owner can flexibly choose to check the change of IDF values, and distribute the new IDF values when these values have changed a lot.

4.6 Parallel Execution of Search

Owing to the tree-based index structure, the proposed search scheme can be executed in parallel, which further improves the search efficiency. For example, we assume there are a set of processors $\mathcal{P} = \{p_1, ..., p_l\}$ available. Given a search request, an idle processor p_i is used to query the root r. If the search could be continued on both the children, and there is an idle processor p_j, the processor p_i continues to deal with one of the children while processor p_j deals with the other one. If there is no idle processor, the current processor is used to deal with the child with larger relevance score, and the other child is put into a waiting queue. Once there is an idle processor, it takes the oldest node in the queue to continue the search. Note that all the processors share the same result list $RList$.

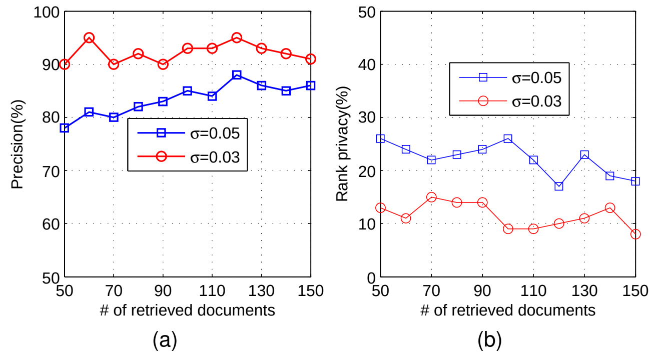


Fig. 4. The precision (a) and rank privacy (b) of searches with different standard deviation σ.

TABLE 2
Precision test of [27]'s basic scheme.

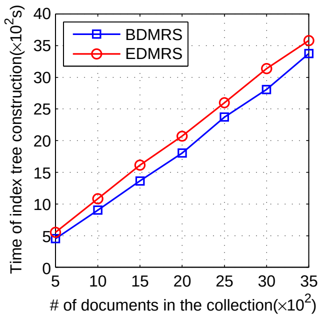
NO	Precision	NO	Precision
1	88%	9	96%
2	94%	10	86.7%
3	97%	11	87.5%
4	100%	12	100%
5	85%	13	82.3%
6	89%	14	100%
7	89%	15	100%
8	96%	16	71.1%

5 Performance Analysis

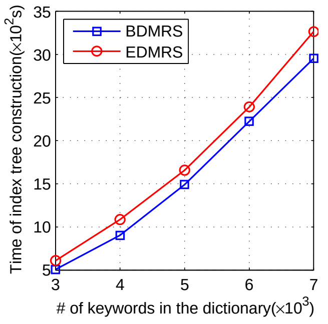
We implement the proposed scheme using C++ language in Windows 7 operation system and test its efficiency on a real-world document collection: the Request for Comments (RFC) [39]. The test includes 1) the search precision on different privacy level, and 2) the efficiency of index construction, trapdoor generation, search, and update. Most of the experimental results are obtained with an Intel Core(TM) Duo Processor (2.93 GHz), except that the efficiency of search is tested on a server with two Intel(R) Xeon(R) CPU E5-2620 Processors (2.0 GHz), which has 12 processor cores and supports 24 parallel threads.

5.1 Precision and Privacy

The search precision of scheme is affected by the dummy keywords in EDMRS scheme. Here, the 'precision' is defined as that in [26]: $P_k = k'/k$, where k' is the number of real top-k documents in the retrieved k documents. If a smaller standard deviation σ is set for the random



(a)



(b)

Fig. 5. Time cost for index tree construction: (a) for the different sizes of document collection with the fixed dictionary, $m = 4000$, and (b) for the different sizes of dictionary with the fixed document collection, $n = 1000$.

TABLE 3
Storage consumption of index tree.

Size of dictionary	1000	2000	3000	4000	5000
BDMRS (MB)	73	146	220	293	367
EDMRS (MB)	95	168	241	315	388

variable $\sum \varepsilon_v$, the EDMRS scheme is supposed to obtain higher precision, and vice versa. The results are shown in Fig. 4(a).

In the EDMRS scheme, phantom terms are added to the index vector to obscure the relevance score calculation, so that the cloud server cannot identify keywords by analyzing the *TF* distributions of special keywords. Here, we quantify the obscureness of the relevance score by "rank privacy", which is defined as:

$$P'_k = \sum |r_i - r'_i|/k^2, \tag{8}$$

where r_i is the rank number of document in the retrieved top-k documents, and r'_i is its real rank number in the whole ranked results. The larger rank privacy denotes the higher security of the scheme, which is illustrated in Fig. 4(b).

In the proposed scheme, data users can accomplish different requirements on search precision and privacy by adjusting the standard deviation σ, which can be treated as a balance parameter.

We compare our schemes with a recent work proposed by Sun *et al.* [27], which achieves high search efficiency. Note that our BDMRS scheme retrieves the search results through exact calculation of document vector and query vector. Thus, top-k search precision of the BDMRS scheme is 100%. But as a similarity-based multi-keyword ranked search scheme, the basic scheme in [27] suffers from precision loss due to the clustering of sub-vectors during index construction. The precision test of [27]'s basic scheme is presented in Table 2. In each test, 5 keywords are randomly chosen as input, and the precision of returned top 100 results is observed. The test is repeated 16 times, and the average precision is 91%.

5.2 Efficiency

5.2.1 *Index Tree Construction*

The process of index tree construction for document collection $\mathcal{F}$ includes two main steps: 1) building an unencrypted KBB tree based on the document collection $\mathcal{F}$, and 2) encrypting the index tree with splitting operation and two multiplications of a $(m \times m)$ matrix. The index structure is constructed following a post order traversal of the tree based on the document collection $\mathcal{F}$, and $O(n)$ nodes are generated during the traversal. For each node, generation of an index vector takes $O(m)$ time, vector splitting process takes $O(m)$ time, and two multiplications of a $(m \times m)$ matrix takes $O(m^2)$ time. As a whole, the time complexity for index tree construction is $O(nm^2)$. Apparently, the time cost for building index tree mainly depends on the cardinality of document collection $\mathcal{F}$ and the number of keywords in dictionary $\mathcal{W}$. Fig. 5 shows that the time cost of index tree construction is almost linear with the size of document collection, and is proportional to the number of keywords in the dictionary. Due to the dimension extension, the index tree construction of EDMRS scheme is slightly more time-consuming than that of BDMRS scheme. Although the index tree construction consumes relatively much time at the data owner side, it is noteworthy that this is a one-time operation.

On the other hand, since the underlying balanced binary tree has space complexity $O(n)$ and every node stores two m-dimensional vectors, the space complexity of the index tree is $O(nm)$. As listed in Table 3, when the document collection is fixed ($n = 1000$), the storage consumption of the index tree is determined by the size of the dictionary.

5.2.2 *Trapdoor Generation*

The generation of a trapdoor incurs a vector splitting operation and two multiplications of a $(m \times m)$ matrix, thus the time complexity is $O(m^2)$, as shown in Fig. 6(a). Typical search requests usually consist of just a few keywords. Fig. 6(b) shows that the number of query keywords has little influence on the overhead of trapdoor generation when the dictionary size is fixed. Due to the dimension extension, the time cost of EDMRS scheme is a little higher than that of BDMRS scheme.

5.2.3 *Search Efficiency*

During the search process, if the relevance score at node u is larger than the minimum relevance score in result list $RList$, the cloud server examines the children of the node; else it returns. Thus, lots of nodes are not accessed during a real search. We denote the number of leaf nodes that contain one or more keywords in the query as θ. Generally, θ is larger than the number of required documents k, but far less than the cardinality of the document collection n. As a balanced binary tree, the height of the index is maintained to be $\log n$, and the complexity of relevance score calculation is $O(m)$. Thus,

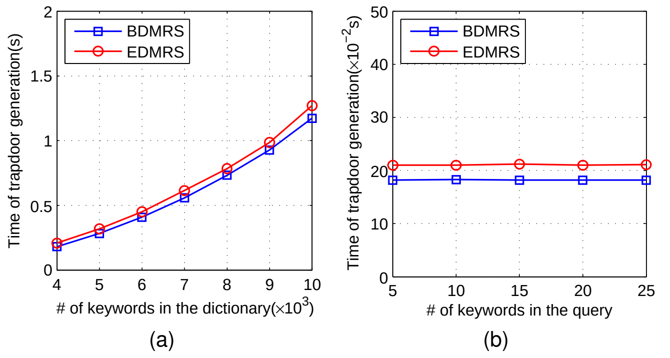


Fig. 6. Time cost for trapdoor generation: (a) for different sizes of dictionary with the fixed number of query keywords, $t = 10$, and (b) for different numbers of query keywords with the fixed dictionary, $m = 4000$.

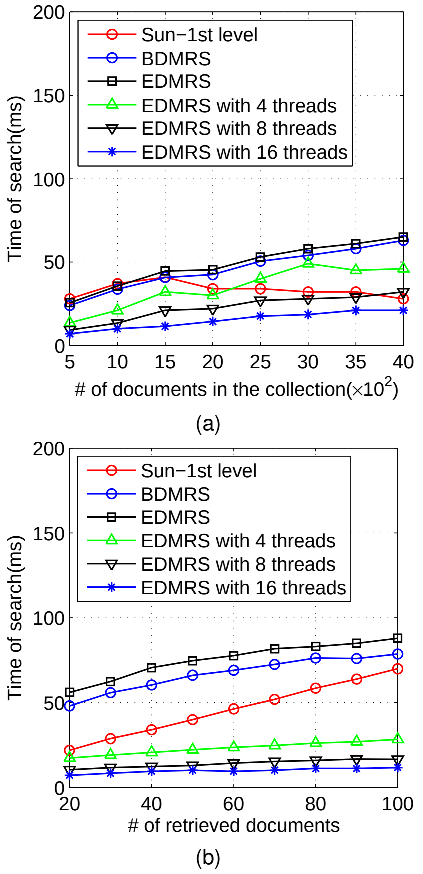


Fig. 7. The efficiency of a search with ten keywords of interest as input: (a) for the different sizes of document collection with the same dictionary, $m = 4000$, and (b) for different numbers of retrieved documents with the same document collection and dictionary, $n = 1000$, and $m = 4000$.

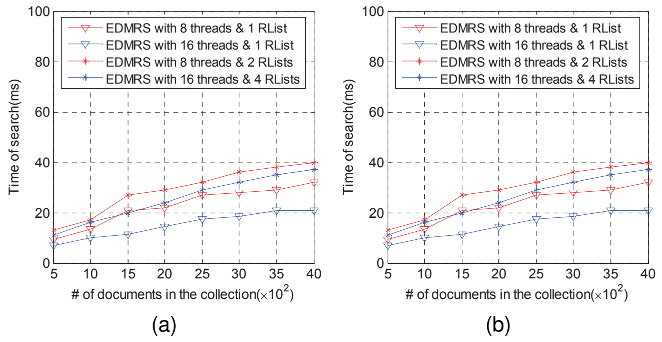


Fig. 8. The efficiency of a search with ten keywords of interest as input: (a) for the different sizes of document collection with the same dictionary, $m = 4000$, and (b) for different numbers of retrieved documents with the same document collection and dictionary, $n = 1000$, and $m = 4000$.

the time complexity of search is $O(\theta m \log n)$. Note that the real search time is less than $\theta m \log n$. It is because 1) many leaf nodes that contain the queried keywords are not visited according to our search algorithm, and 2) the accessing paths of some different leaf nodes share the mutual traversed parts. In addition, the parallel execution of search process can increase the efficiency a lot.

We test the search efficiency of the proposed scheme on a server which supports 24 parallel threads. The search performance is tested respectively by starting 1, 4, 8 and 16 threads. We compare the search efficiency of our scheme with that of Sun *et al.* [27]. In the implementation of Sun's code, we divide 4000 keywords into 50 levels. Thus, each level contains 80 keywords. According to [27], the higher level the query keywords reside, the higher the search efficiency is. In our experiment, we choose ten keywords from the 1st level (the highest level, the optimal case) for search efficiency comparison. Fig. 7 shows that if the query keywords are chosen from the 1st level, our scheme obtains almost the same efficiency as [27] when we start 4 threads.

Fig. 7 also shows that the search efficiency of our scheme increases a lot when we increase the number of threads from 1 to 4. However, when we continue to increase the threads, the search efficiency is not increased remarkably. Our search algorithm can be executed in parallel to improve the search efficiency. But all the started threads will share one result list $RList$ in mutually exclusive manner. When we start too many threads, the threads will spend a lot of time for waiting to read and write the $RList$.

An intuitive method to handle this problem is to construct multiple result lists. However, in our scheme, it will not help to improve the search efficiency a lot. It is because that we need to find k results for each result list and time complexity for retrieving each result list is $O(\theta m \log n/l)$. In this case, the multiple threads will not save much time, and selecting k results from the multiple result list will further increase the time consumption. In the Fig. 8, we show the time consumption when we start multiple threads with multiple result lists. The experimental results prove that our scheme will obtain better search efficiency when we start multiple threads with only one result list.

5.2.4 Update Efficiency

In order to update a leaf node, the data owner needs to update $\log n$ nodes. Since it involves an encryption operation for index vector at each node, which takes $O(m^2)$ time, the time complexity of update operation is thus $O(m^2 \log n)$. We illustrate the time cost for the

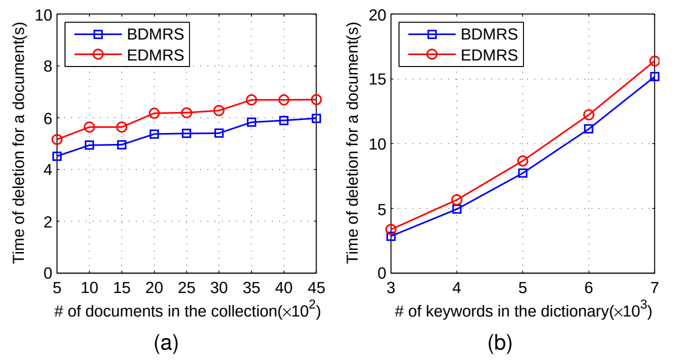


Fig. 9. Time cost for deletion of a document: (a) for the different sizes of document collection with the same dictionary, $m = 4000$, and (b) for the same document collection with different sizes of dictionary, $n = 1000$.

deletion of a document. Fig. 9(a) shows that when the size of dictionary is fixed, the deletion of a document takes nearly logarithmic time with the size of document collection. And Fig. 9(b) shows that the update time is proportional to the size of dictionary when the document collection is fixed.

In addition, the space complexity of each node is $O(m)$. Thus, space complexity of the communication package of updating a document is $O(m \log n)$.

6 Conclusion and Future work

In this paper, a secure, efficient and dynamic search scheme is proposed, which supports not only the accurate multi-keyword ranked search but also the dynamic deletion and insertion of documents. We construct a special keyword balanced binary tree as the index, and propose a "Greedy Depth-first Search" algorithm to obtain better efficiency than linear search. In addition, the parallel search process can be carried out to further reduce the time cost. The security of the scheme is protected against two threat models by using the secure kNN algorithm. Experimental results demonstrate the efficiency of our proposed scheme.

There are still many challenge problems in symmetric SE schemes. In the proposed scheme, the data owner is responsible for generating updating information and sending them to the cloud server. Thus, the data owner needs to store the unencrypted index tree and the information that are necessary to recalculate the IDF values. Such an active data owner may not be very suitable for the cloud computing model. It could be a meaningful but difficult future work to design a dynamic searchable encryption scheme whose updating operation can be completed by cloud server only, meanwhile reserving the ability to support multi-keyword ranked search. In addition, as the most of works about searchable encryption, our scheme mainly considers the challenge from the cloud server. Actually, there are many secure challenges in a multi-user scheme. Firstly, all the users usually keep the same secure key for trapdoor generation in a symmetric SE scheme. In this case, the revocation of the user is big challenge. If it is needed to revoke a user in this scheme, we need to rebuild the index and distribute the new secure keys to all the authorized users. Secondly, symmetric SE schemes usually assume that all the data users are trustworthy. It is not practical and a dishonest data user will lead to many secure problems. For example, a dishonest data user may search the documents and distribute the decrypted documents to the unauthorized ones. Even more, a dishonest data user may distribute his/her secure keys to the unauthorized ones. In the future works, we will try to improve the SE scheme to handle these challenge problems.

Acknowledgments

This work is supported by the NSFC (61173141, 61232016, 61173142, 61173136, 61103215, 61373132, 61373133, 61300237, 61373167), GYHY201206033, 201301030, 2013DFG12860, BC2013012, Fund of Jiangsu Engineering Center of Network Monitoring (KJR1308, KJR1402), Fund of MOE Internet Innovation Platform (KJRP1403), Natural Science Foundation of Hubei Province (2013CFB297) and PAPD fund.

The authors would like to thank Wenhai Sun and Bing Wang to provide us with their experimental code for performance comparisons, and thank the editor and reviewers for their valuable suggestions.

References

[1] K. Ren, C. Wang, Q. Wang *et al.*, "Security challenges for the public cloud," *IEEE Internet Computing*, vol. 16, no. 1, pp. 69–73, 2012.

[2] S. Kamara and K. Lauter, "Cryptographic cloud storage," in *Financial Cryptography and Data Security*. Springer, 2010, pp. 136–149.

[3] C. Gentry, "A fully homomorphic encryption scheme," Ph.D. dissertation, Stanford University, 2009.

[4] O. Goldreich and R. Ostrovsky, "Software protection and simulation on oblivious rams," *Journal of the ACM (JACM)*, vol. 43, no. 3, pp. 431–473, 1996.

[5] D. Boneh, G. Di Crescenzo, R. Ostrovsky, and G. Persiano, "Public key encryption with keyword search," in *Advances in Cryptology-Eurocrypt 2004*. Springer, 2004, pp. 506–522.

[6] D. Boneh, E. Kushilevitz, R. Ostrovsky, and W. E. Skeith III, "Public key encryption that allows pir queries," in *Advances in Cryptology-CRYPTO 2007*. Springer, 2007, pp. 50–67.

[7] D. X. Song, D. Wagner, and A. Perrig, "Practical techniques for searches on encrypted data," in *Security and Privacy, 2000. S&P 2000. Proceedings. 2000 IEEE Symposium on*. IEEE, 2000, pp. 44–55.

[8] E.-J. Goh *et al.*, "Secure indexes." *IACR Cryptology ePrint Archive*, vol. 2003, p. 216, 2003.

[9] Y.-C. Chang and M. Mitzenmacher, "Privacy preserving keyword searches on remote encrypted data," in *Proceedings of the Third international conference on Applied Cryptography and Network Security*. Springer-Verlag, 2005, pp. 442–455.

[10] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky, "Searchable symmetric encryption: improved definitions and efficient constructions," in *Proceedings of the 13th ACM conference on Computer and communications security*. ACM, 2006, pp. 79–88.

[11] J. Li, Q. Wang, C. Wang, N. Cao, K. Ren, and W. Lou, "Fuzzy keyword search over encrypted data in cloud computing," in *INFOCOM, 2010 Proceedings IEEE*. IEEE, 2010, pp. 1–5.

[12] M. Kuzu, M. S. Islam, and M. Kantarcioglu, "Efficient similarity search over encrypted data," in *Data Engineering (ICDE), 2012 IEEE 28th International Conference on*. IEEE, 2012, pp. 1156–1167.

[13] C. Wang, K. Ren, S. Yu, and K. M. R. Urs, "Achieving usable and privacy-assured similarity search over outsourced cloud data," in *INFOCOM, 2012 Proceedings IEEE*. IEEE, 2012, pp. 451–459.

[14] B. Wang, S. Yu, W. Lou, and Y. T. Hou, "Privacy-preserving multi-keyword fuzzy search over encrypted data in the cloud," in *IEEE INFOCOM*, 2014.
[15] P. Golle, J. Staddon, and B. Waters, "Secure conjunctive keyword search over encrypted data," in *Applied Cryptography and Network Security*. Springer, 2004, pp. 31–45.
[16] Y. H. Hwang and P. J. Lee, "Public key encryption with conjunctive keyword search and its extension to a multi-user system," in *Proceedings of the First international conference on Pairing-Based Cryptography*. Springer-Verlag, 2007, pp. 2–22.
[17] L. Ballard, S. Kamara, and F. Monrose, "Achieving efficient conjunctive keyword searches over encrypted data," in *Proceedings of the 7th international conference on Information and Communications Security*. Springer-Verlag, 2005, pp. 414–426.
[18] D. Boneh and B. Waters, "Conjunctive, subset, and range queries on encrypted data," in *Proceedings of the 4th conference on Theory of cryptography*. Springer-Verlag, 2007, pp. 535–554.
[19] B. Zhang and F. Zhang, "An efficient public key encryption with conjunctive-subset keywords search," *Journal of Network and Computer Applications*, vol. 34, no. 1, pp. 262–267, 2011.
[20] J. Katz, A. Sahai, and B. Waters, "Predicate encryption supporting disjunctions, polynomial equations, and inner products," in *Advances in Cryptology–EUROCRYPT 2008*. Springer, 2008, pp. 146–162.
[21] E. Shen, E. Shi, and B. Waters, "Predicate privacy in encryption systems," in *Proceedings of the 6th Theory of Cryptography Conference on Theory of Cryptography*. Springer-Verlag, 2009, pp. 457–473.
[22] A. Lewko, T. Okamoto, A. Sahai, K. Takashima, and B. Waters, "Fully secure functional encryption: attribute-based encryption and (hierarchical) inner product encryption," in *Proceedings of the 29th Annual international conference on Theory and Applications of Cryptographic Techniques*. Springer-Verlag, 2010, pp. 62–91.
[23] A. Swaminathan, Y. Mao, G.-M. Su, H. Gou, A. L. Varna, S. He, M. Wu, and D. W. Oard, "Confidentiality-preserving rank-ordered search," in *Proceedings of the 2007 ACM workshop on Storage security and survivability*. ACM, 2007, pp. 7–12.
[24] S. Zerr, D. Olmedilla, W. Nejdl, and W. Siberski, "Zerber+ r: Top-k retrieval from a confidential index," in *Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology*. ACM, 2009, pp. 439–449.
[25] C. Wang, N. Cao, K. Ren, and W. Lou, "Enabling secure and efficient ranked keyword search over outsourced cloud data," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 23, no. 8, pp. 1467–1479, 2012.
[26] N. Cao, C. Wang, M. Li, K. Ren, and W. Lou, "Privacy-preserving multi-keyword ranked search over encrypted cloud data," in *IEEE INFOCOM*, April 2011, pp. 829–837.
[27] W. Sun, B. Wang, N. Cao, M. Li, W. Lou, Y. T. Hou, and H. Li, "Privacy-preserving multi-keyword text search in the cloud supporting similarity-based ranking," in *Proceedings of the 8th ACM SIGSAC symposium on Information, computer and communications security*. ACM, 2013, pp. 71–82.
[28] C. Orencik, M. Kantarcioglu, and E. Savas, "A practical and secure multi-keyword search method over encrypted cloud data," in *Cloud Computing (CLOUD), 2013 IEEE Sixth International Conference on*. IEEE, 2013, pp. 390–397.
[29] W. Zhang, S. Xiao, Y. Lin, T. Zhou, and S. Zhou, "Secure ranked multi-keyword search for multiple data owners in cloud computing," in *Dependable Systems and Networks (DSN), 2014 44th Annual IEEE/IFIP International Conference on*. IEEE, 2014, pp. 276–286.
[30] S. Kamara, C. Papamanthou, and T. Roeder, "Dynamic searchable symmetric encryption," in *Proceedings of the 2012 ACM conference on Computer and communications security*. ACM, 2012, pp. 965–976.
[31] S. Kamara and C. Papamanthou, "Parallel and dynamic searchable symmetric encryption," in *Financial Cryptography and Data Security*. Springer, 2013, pp. 258–274.
[32] D. Cash, S. Jarecki, C. Jutla, H. Krawczyk, M.-C. Roşu, and M. Steiner, "Highly-scalable searchable symmetric encryption with support for boolean queries," in *Advances in Cryptology–CRYPTO 2013*. Springer, 2013, pp. 353–373.
[33] D. Cash, J. Jaeger, S. Jarecki, C. Jutla, H. Krawczyk, M.-C. Rosu, and M. Steiner, "Dynamic searchable encryption in very large databases: Data structures and implementation," in *Proc. of NDSS*, vol. 14, 2014.
[34] C. D. Manning, P. Raghavan, and H. Schütze, *Introduction to information retrieval*. Cambridge university press Cambridge, 2008, vol. 1.
[35] B. Gu and V. S. Sheng, "Feasibility and finite convergence analysis for accurate on-line -support vector learning," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 24, no. 8, pp. 1304–1315, 2013.
[36] X. Wen, L. Shao, W. Fang, and Y. Xue, "Efficient feature selection and classification for vehicle detection."
[37] H. Delfs and H. Knebl, *Introduction to cryptography: principles and applications*. Springer, 2007.
[38] W. K. Wong, D. W.-l. Cheung, B. Kao, and N. Mamoulis, "Secure knn computation on encrypted databases," in *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*. ACM, 2009, pp. 139–152.
[39] "Request for comments," http://www.rfc-editor.org/index.html.

Zhihua Xia received his BE in Hunan City University, China, in 2006, PhD in computer science and technology from Hunan University, China, in 2011. He works as a lecturer in School of Computer & Software, Nanjing University of Information Science & Technology. His research interests include cloud computing security, and digital forensic.

Xinhui Wang received her BE in software engineering from Nanjing University of Information Science & Technology in 2012, China. She is currently pursuing her MS in computer science and technology at the College of Computer and Software, in Nanjing University of Information Science & Technology, China. Her research interest is cloud computing security.

Xingming Sun received his BS in mathematics from Hunan Normal University, China, in 1984, MS in computing science from Dalian University of Science and Technology, China, in 1988, and PhD in computing science from Fudan University, China, in 2001. He is currently a professor in School of Computer & Software, Nanjing University of Information Science & Technology, China. His research interests include network and information security, digital watermarking, and data security in cloud.

Qian Wang received his BS degree from Wuhan University, China, in 2003, his MS degree from Shanghai Institute of Microsystem and Information Technology, Chinese Academy of Sciences, China, in 2006, and his PhD degree from Illinois Institute of Technology in 2012, all in electrical and computer engineering. Currently, he is an associate professor with the School of Computer Science, Wuhan University. His research interests include wireless network security and privacy, cloud computing security, and applied cryptography. He is a corecipient of the Best Paper Award from IEEE ICNP 2011. He is a member of the ACM and a member of the IEEE.