International Journal of Security and Its Applications
Vol.8, No.1 (2014), pp.319-332
http://dx.doi.org/10.14257/ijsia.2014.8.1.30

Dynamic Multi-keyword Top-k Ranked Search over Encrypted Cloud Data

Xingming Sun, Xinhui Wang, Zhihua Xia, Zhangjie Fu and Tao Li

Jiangsu Engineering Center of Network Monitoring, Nanjing University of Information Science & Technology, Nanjing, 210044, China
sunnudt@163.com, wxh_nuist@163.com, xia_zhihua@163.com, wwwfzj@126.com, taoleehome@hotmail.com

Abstract

Nowadays, more and more people are motivated to outsource their local data to public cloud servers for great convenience and reduced costs in data management. But in consideration of privacy issues, sensitive data should be encrypted before outsourcing, which obsoletes traditional data utilization like keyword-based document retrieval. In this paper, we present a secure and efficient multi-keyword ranked search scheme over encrypted data, which additionally supports dynamic update operations like deletion and insertion of documents. Specifically, we construct an index tree based on vector space model to provide multi-keyword search, which meanwhile supports flexible update operations. Besides, cosine similarity measure is utilized to support accurate ranking for search result. To improve search efficiency, we further propose a search algorithm based on "Greedy Depth-first Traverse Strategy". Moreover, to protect the search privacy, we propose a secure scheme to meet various privacy requirements in the known ciphertext threat model. Experiments on the real-word dataset show the effectiveness and efficiency of proposed scheme.

Keywords: *Encrypted cloud data, Multi-keyword ranked search, Dynamic update*

1. Introduction

Recent years, cloud computing enjoys great reputation in data management due to its outstanding capability in computing, storage and various applications. Through cloud services, people could enjoy convenient, on-demand network access to a shared pool of configurable computing resources with great efficiency and minimal economic overhead [1]. Despite of the various advantages offered by cloud services, transfer of sensitive information (such as e-mails, company finance data, and government documents, etc) to semi-trusted cloud server brings concerns about privacy issues. For instance, the cloud server may leak information to unauthorized entities or even be hacked, which puts the outsourced data at risk. Traditionally, sensitive data should be encrypted by data owners before outsourcing, which, however, obsoletes traditional data utilization service like keyword-based information retrieval.

To enable traditional utilization on encrypted data, searchable encryption techniques [2-14], especially those based on symmetric key cryptography [4-14], are proposed for efficient keyword search over encrypted data. By now, many symmetric searchable encryption (SSE) schemes have been developed as an attempt for enriching the search flexibility, like single-keyword search [4-10] and multi-keyword search [11-14]. To enhance the accuracy of search result, multi-keyword search is more suitable for real world than single-keyword search. Among those multi-keyword search works, many have realized the conjunctive keyword

ISSN: 1738-9976 IJSIA

search, subnet search, or range search [11, 12], but they don't support accurate ranked search. In plaintext information retrieval (IR) community, there are many state-of-the-art technologies for multi-keyword ranked search, for instance, cosine measure in the vector space model [15]. To provide ranked search functionality on encrypted data, Cao et al. [13] proposed a privacy-preserving multi-keyword ranked search scheme. With "coordinate matching", search result is ranked according to the number of matched keywords, which is not accurate enough. And their search complexity is linear with the number of documents in dataset. Then, in [14], Sun et al. proposed a multi-keyword search scheme that supports similarity-based ranking. They constructed a searchable index tree based on vector space model and adopted cosine measure together with "term frequency (TF) × inverse document frequency (IDF)" weight to provide accurate ranking. Finally, their search algorithm achieves better-than-linear search efficiency.

However, most of the above schemes work for static data, and can not efficiently handle dynamic collections (*i.e.*, document collections that must be updated). Based on the inverted index structure proposed in [8], Kamara *et al.*, [16] constructed an encrypted inverted index that can handle dynamic data efficiently. In their index structure, same documents in different index entries are linked by pointers. Additionally, two arrays are constructed for tracing documents for a specific keyword and tracing keywords for a specific document, by which insertion or deletion of a document could be realized. But, their scheme is very complex and difficult to implement. Subsequently, as an improvement, Kamara *et al.*, [17] introduced a new tree-based scheme that can simply handle dynamic data. In their scheme, every document has a corresponding index, and all these indexes are merged to a hierarchical index tree. Thus, update operations could be easily implemented due to the inherent feature of tree structure. Besides, the search scheme can achieve sub-linear search time through parallel execution. However, this scheme is designed for single-keyword search and doesn't take the accurate result ranking into account.

In this paper, we propose a secure dynamic multi-keyword ranked search scheme over encrypted cloud data, which supports top-k retrieval and dynamic updates on dataset. Specifically, we adopt the vector space model to provide multi-keyword queries, and cosine measure together with TF×IDF weight is utilized to achieve accurate ranked results. To improve the search efficiency, we construct a tree-based index structure and propose a top-k ranked search algorithm over this index which has logarithmic search time. Besides, benefiting from the index tree structure, update on documents is available in our scheme. The proposed dynamic multi-keyword ranked search scheme (DMRS) is secure under the known ciphertext model. Our contributions are summarized as follows:

1) Our proposed search scheme achieves multi-keyword ranked search over encrypted data with high efficiency and search result accuracy.

2) We propose a secure DMRS scheme which meets privacy requirements in the known ciphertext model.

3) Benefiting from tree-based index structure, our search scheme supports dynamic update operation (like deletion and insertion) on documents, which caters to real-world needs and is superior to most current static schemes.

The reminder of this paper is organized as follows. In Section 2, we give a brief introduction to the system model and threat model, our design goals, and the preliminary. Section 3 describes the DMRS search scheme in detail, and simulation results and performance analysis are presented in the section that follows. We conclude the paper in Section 5.

2. Problem Formulation

2.1. The System and Threat Model

The system model in this paper involves three different entities: the data owner, the data user, and the cloud server, as illustrated in Figure 1. The data owner outsources his local data to the cloud server, including a collection of encrypted documents C generated from F, and an encrypted searchable index tree I. Through access control mechanism [18], which is a separate issue out of the scope of this paper, a user could be authorized to access the data of data owner. Then, with t query keywords, the authorized user would generate a corresponding trapdoor T for the search request through search control mechanisms. Upon receiving the trapdoor T, the cloud server executes similarity search over the index tree I and finally returns the corresponding set of ranked encrypted documents. Moreover, user could choose the number of retrieved documents through submitting parameter k, which means the cloud server could terminate searching process while the top-k documents have been retrieved. This would simultaneously reduce the communication overhead and satisfy data users' requirements. Finally, the authorized user decrypts these received documents.

The cloud server in this paper is considered as "honest-but-curious". Specifically, the cloud server honestly and correctly executes instructions according to the designated protocol. Meanwhile, it is "curious" to infer and analyze received data (including index and message flows) in its storage, which helps it acquire additional information.

Known ciphertext threat model: In this model, the cloud server only knows the outsourced data from the data owner (including encrypted document set C and the searchable index tree I), and the encrypted query (trapdoor) T submitted by authorized user.

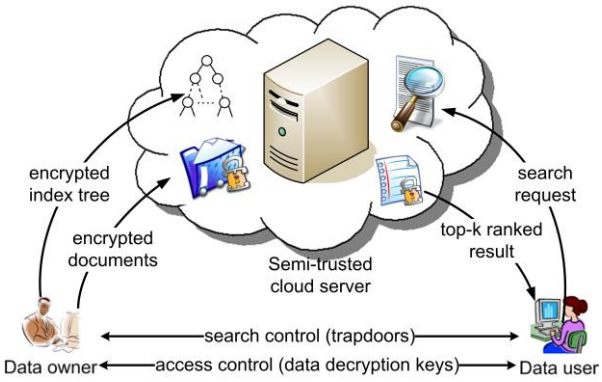


Figure 1. Architecture of the Ranked Search over Encrypted Cloud Data

2.2. Design Goals

To enable efficient, secure and dynamic multi-keyword ranked search over outsourced encrypted cloud data under the aforementioned models, our system design should simultaneously achieve the following design goals.

1) Dynamic Multi-Keyword Ranked Search: To design a search scheme over encrypted data which provides not only effective multi-keyword query and accurate result ranking, but also dynamic update on document collections.

2) Search Efficiency: Our search scheme aims to achieve better practical search efficiency than linear search [13] by exploring a tree-based index structure and an efficient search algorithm.

3) Privacy-preserving: To prevent the cloud server from learning additional information from the dataset, the index tree, and the queries. The specific search privacy requirements are summarized as follows, *1) Index Confidentiality and Query Confidentiality*: the underlying plaintext information (including keywords in the index and query, keywords' TF values 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 dataset.

2.3. Notations and Preliminaries

- F – the plaintext document collection, denoted as a set of n documents $F = (f_1, f_2, \ldots, f_n)$.
- C – the encrypted document collection for F, denoted as $C = (c_1, c_2, \ldots, c_n)$.
- W – the dictionary, *i.e.*, the keyword set consisting of m keywords, denoted as $W = (w_1, w_2, \ldots, w_m)$.
- I – the searchable encrypted index tree.
- T – the unencrypted form of index tree I, which is stored in data owner side.
- D_u – the index vector stored in node u of index tree.
- Q – the query vector generated from search request.
- I_u – the encrypted form of D_u.
- T – the encrypted form of Q, which is always called the trapdoor for the search request.

Vector space model and ranking function: Vector space model [15] is widely used in plaintext information retrieval, which efficiently represents documents with multi-dimensional vectors. And in this paper, we use the cosine measure together with TF×IDF rule to provide accurate ranking. Here, term frequency (TF) is simply the number of times a given term or keyword appears within a document, and inverse document frequency (IDF) is obtained through dividing the number of documents in the whole collection by the number of documents containing the term. For document vector, each element represents the TF value of corresponding keyword in this document, and for query vector, each element represents the IDF value of corresponding keyword in the dataset. To quantify the similarity of each document vector and the query vector, the deviation of angles (*i.e.*, cosine values) between the two vectors is calculated. There are various similarity evaluation functions for cosine measure, and we choose the one in [15]. Set d_i as the document vector of document f_i and q as the query vector, the similarity score (cosine value) of the document and query is calculated as follows:

$$Cos(d_i, q) = \frac{d_i \times q}{|d_i||q|} = \frac{1}{|d_i||q|} \sum_{t \in d_i \cap q} tf_{i,t} \times idf_t \quad (1)$$

where $tf_{i,t} = (1 + \ln N_{i,t}), idf_t = \ln\left(1 + \frac{N}{N_t}\right)$.

Here, $tf_{i,t}$ denotes the TF value of term t in document f_i, idf_t denotes the IDF value of term t in dataset, $N_{i,t}$ denotes the number of term t in document f_i, N_t denotes the number of documents that contain term t, N denotes the total number of documents in the collection,

and $|d_i|, |q|$ are the Euclid lengths of vector d_i and q, functioning as the normalization factors. Then the value of cosine measure can be denoted as the inner product of d_i and q 's unit vectors.

Keyword red-black tree-based search algorithm: The red-black tree [19] is a type of self-balancing binary search tree which supports efficient search and update operations. Adapted from red-black tree, the keyword red-black (KRB) tree proposed in [17] is a dynamic data structure that can efficiently answer multi-keyword queries. The data structure of node u in KRB tree could be simply defined as $\{D_u, id, v, z\}$, where v is u 's left child and z is u 's right child. Note that only the leaf node has the real value of id that points to a specific document. D_u is the m -bit binary index vector of node u, and $D_u[i]$ represents keyword w_i (i = 1, ... , m). If u is a leaf node, $D_u[i] = 1$ if and only if the document pointed by u ® id (i.e. f_{id}) contains keyword w_i, otherwise, $D_u[i] = 0$; if u is an internal node, $D_u[i] = D_v[i] \vee D_z[i]$, i.e. D_u is computed by the bitwise Boolean OR operation of children' index vectors. Then, we can say that if $D_u[i] = 1$ for internal node u, there is at least one path from u to some leaf storing identifier id that f_{id} contains w_i. In this paper, we modify the KRB tree and name it MKRB (Modified KRB) tree. The detailed construction procedure will be illustrated in Section 3, which is denoted as $buildIndexTree(F)$.

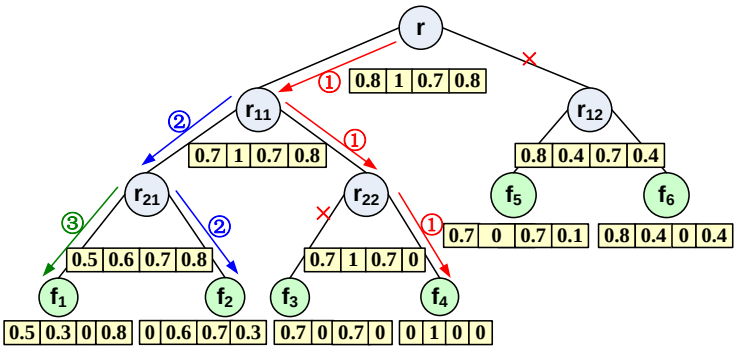


Figure 2. The Tree-based Index Structure for a Collection of n=6 Documents Indexed over m=4 Keywords

3. DMRS Scheme

3.1. Tree-based Index Construction

In this part, we present a detailed description of our index tree construction. The procedure is presented in Table 1, and an example of our index tree is shown in Figure 2. Note that the index tree T built here is plaintext.

Table 1. The Process of Index Tree Construction

buildIndexTree(F)

1. Initialization:

For input document set F =($f_1, f_2, \ldots, f_n$), which is imposed by the ordering of the identifiers id =(1,2, ... ,n), build a red-black tree T on top of id (1,2, ... ,n). The node's data structure is defined as the same as that in KRB tree: $\{D_u, id, v, z\}$

2. Add data to all nodes:

- If u is a leaf node, $D_u[i] = tf_{id,i}$ if and only if the document pointed by u ® id (i.e. f_{id}) contains keyword w_i ($tf_{id,i}$ is the normalized TF value of keyword i in document f_{id}), otherwise, $D_u[i] = 0$.
- If u is an internal node, with left child v and right child z , D_u is computed recursively as follows:

$$D_u[i] = \max(D_v[i], D_z[i]), i = (1,2,\ldots,m) \quad (2)$$

Specifically, $D_u[i]$ stores the biggest normalized TF value of w_i among its child nodes. If $D_u[i]$ ¹ 0 , then there is at least one path from u to some leaf that stores the identifier id , such that f_{id} contains w_i . Meanwhile, we can get the biggest normalized TF value of keyword w_i to those documents in the subtree rooted by u , which can be utilized to calculate the maximum possible similarity score in these documents. For example, through node r_{11} in Figure 2, we know that the biggest normalized TF value of the second keyword among $\{f_1, f_2, f_3, f_4\}$ is 1.

3. Output plaintext index tree T .

3.2. Search Algorithm

The search process of our DMRS scheme starts from the root node with a recursive procedure upon the tree in a special depth-first manner, which is called as "Greedy Depth-first Traverse Strategy". Specifically, if the node's similarity score is less than or equal to the minimum similarity score of the currently selected top-k documents, search process returns to the parent node, otherwise, it goes down to examine the child node. The similarity score of each node u is calculated as Formula (1), *i.e.*, the inner product of query vector Q and data vector D_u . This procedure is executed recursively until the objects with top-k scores are selected. The search can be done very efficiently, since only part of the index tree is visited due to the relatively accurate maximum score prediction. Algorithm 1 shows the process of our proposed search scheme.

The following notations are used in the pseudo code for our tree-based search algorithm:

- SimScore(D_u, Q) – function of similarity score calculation for unit query vector Q and unit index vector D_u , which is equal to $Cos(D_u, Q)$.
- $Top\text{-}k\_List$ – the list of the top-k documents' identifies which are arranged in descending order according to documents' similarity scores.
- k^{th}_score – the current lowest score of top-k documents in $Top\text{-}k\_List$, and is set as 0

when the size of $Top\text{-}k\_List$ is less than k.

- *hchild* – child node with higher similarity score.
- *lchild* – child node with lower similarity score.

Algorithm 1 Proposed Search Algorithm on index tree I

```
procedure GreedyDFS( IndexTreeNode u )
if ( u is not a leaf node) then
        if ( SimScore(D_u,Q) > k^th_score ) then
                GreedyDFS(u → hchild);
                GreedyDFS(u → lchild);
        else return;
        end if
else Update(u);
end if
end procedure

procedure Update( IndexTreeNode l )
if (the size of Top-k_List is less than k ) then
        add l → id to Top-k_List ;
else
        if ( SimScore(D_l,Q) > k^th_score ) then
                delete the identifier with the score as k^th_score from Top-k_List and
                insert l → id ;
                refresh k^th_score ;
        else return;
        end if
end if
end procedure
```

Figure 2 shows an example of our search scheme. The arrows with numbers indicate the search process for top-3 documents while query $Q = (0, 0.92, 0, 0.38)$ (note that the index vectors of leaf nodes have been normalized): the arrows with ① indicate the first search round and finally obtain document f_4 with similarity score 0.92; the arrows with ② indicate the search for second document and obtain document f_2 with score 0.67; and the arrow with ③ indicates the search for third document and finally retrieves document f_1 with score 0.58 as result. The cross means this path can not lead to reasonable results. For example, the path from r to r_{12} is rejected as the similarity score of r_{12} is less than the minimum similarity score of $\{f_4, f_2, f_1\}$, *i.e.*, 0.58.

3.3. Secure Scheme

As soon as the plaintext index tree is built, secure encryption scheme needs to be executed to prevent information leakage. We adopt the encryption scheme in [14] to secure our index tree, the whole process is described as follows:

- **Setup :** Initially, the data owner produces the secret key SK, including: 1) a randomly generated vector S of m-bit (m is the size of dictionary); 2) two $(m \times m)$ invertible matrices $\{M_1, M_2\}$. And SK is composed of $\{S, M_1, M_2\}$.
- **GenIndex**(F, SK) **:** Firstly, unencrypted index tree T is built on F according to $buildIndexTree(\mathrm{F})$. Then, for every index vector D_u stored in node u of T, the data owner generates two random vectors $\{D_u', D_u''\}$ from D_u. Specifically, if $S[i] = 0$, $D_u[i] = D_u'[i] = D_u''[i]$; if $S[i] = 1$, $D_u'[i]$ and $D_u''[i]$ are set to two random values whose sum equals to $D_u[i]$, i.e. $D_u[i] = D_u'[i] + D_u''[i]$. Finally, the encrypted index tree I is built where every node u stores an encrypted index vector $I_u = \{M_1^T D_u', M_2^T D_u''\}$.
- **GenTrapdoor**$(\mathbb{W}, SK)$ **:** With keywords of interest in $\mathbb{W}$ as input, the query vector Q is generated where each dimension stores a normalized IDF value. Similarly, Q is split into two random vectors as $\{Q', Q''\}$, the difference is that if $S[i] = 0$, $Q'[i]$ and $Q''[i]$ are set to two random values whose sum equals to $Q[i]$, i.e. $Q[i] = Q'[i] + Q''[i]$; otherwise, $Q[i] = Q'[i] = Q''[i]$. Finally, the trapdoor T is generated as $\{M_1^{-1}Q', M_2^{-1}Q''\}$.
- **SimEvaluation**(I_u, T) **:** With the trapdoor T, the cloud server computes the similarity scores of nodes visited in the index tree I, as shown in equation (3). Note that only the leaf nodes obtain the real similarity scores of documents.

$$\begin{aligned} & Cos(I_u, T) \\ &= \{M_1^T D_u', M_2^T D_u''\} \times \{M_1^{-1}Q', M_2^{-1}Q''\} \\ &= D_u' \times Q' + D_u'' \times Q'' \\ &= D_u \times Q \\ &= Cos(D_u, Q) \end{aligned} \tag{3}$$

Security analysis We analyze the security of DMRS scheme from three aspects of privacy requirements as described in section 2.

1) Index confidentiality and query confidentiality: In DMRS, I_u and T are obfuscated vectors, which means the cloud server can not infer the original vectors D_u or Q without the secret key SK. Therefore, index confidentiality and query confidentiality are well protected.

2) Query unlinkability: The trapdoor of query vector is generated from random splitting operation, which means same search requests would be transformed into different query vectors (trapdoors), therefore, the query unlinkability is protected. However, equipped with capability on tracking visited nodes with corresponding similarity scores, the cloud server might be able to link the same search requests according to the same similarity scores. Under this circumstance, the query unlinkability is unavailable.

3) Keyword privacy: In the known ciphertext model, the cloud server is supposed to only know the encrypted document set C, index tree I and trapdoor T. Therefore, without other background information, the cloud server is unable to deduce keywords or TF/IDF values from the result similarity scores, as is proven in [20]. However, in enhanced threat model, the cloud server may be equipped with more knowledge like document/keyword frequency statistics of the dataset. Then the cloud server could launch statistical attack to deduce or even identify specific keywords in the query [10].

As an improvement, our future work aims to design a secure scheme that meets all the privacy requirements above even in enhanced threat model.

3.4. Dynamic Update Operation

Since our DMRS scheme is designed on a red-black tree data structure, the dynamic operations (like insertion or deletion of a document) could be executed efficiently through structural update on the index tree. Furthermore, since the documents are directly related to the leaf nodes, the whole structure of index tree would change little. The specific process is presented as follows:

- **GenInfo**$(i, ud, \mathrm{T}, \mathrm{C})$**:** In this phase, the data owner generates the update information ud (either an insertion or a deletion) for document f_i, and output $\mathrm{info}_{i,ud}$. Since the documents are directly related to the leaf nodes, the update operation is equivalent to insertion or deletion of a leaf node. Thus, only a path from root to leaf needs to be accessed during a specific update for a document. We define this path as P. After structural update on T, P 's information is updated and stored in $\mathrm{info}_{i,ud}$. For instance, in Figure 2, $\mathrm{info}_{4,ud}$, where ud is "deletion of file f_4", contains the path composed of nodes r_{22}, r_{11} and r. Note that the update is based on document identifies, which doesn't access the actual content of documents.
- **EncInfo**$(SK, f_i, \mathrm{info}_{i,ud})$**:** According to the update information $\mathrm{info}_{i,ud}$, the data owner encrypts the portion P and outputs the final updated data $(\mathrm{P}\not\subset, \mathrm{C}_i)$. If the update ud is an insertion of document f_i, the encrypted form C_i is generated for the added document f_i firstly. Secondly, P is encrypted by SK and turned to P¢as the same encryption process as that in **GenIndex**(F, SK). If the update ud is a deletion of document f_i, then C_i means the document to be removed from cloud server.
- **Update**$(\mathrm{I}, \mathrm{C}, \mathrm{P}\not\subset, \mathrm{C}_i)$**:** On input $(\mathrm{P}\not\subset, \mathrm{C}_i)$, the cloud server just covers the original data of index tree I with the new data P¢and outputs the new encrypted index I ¢with the new set of ciphertexts C¢. Since P ¢merely contains nodes of a path, little part of the original index I is changed.

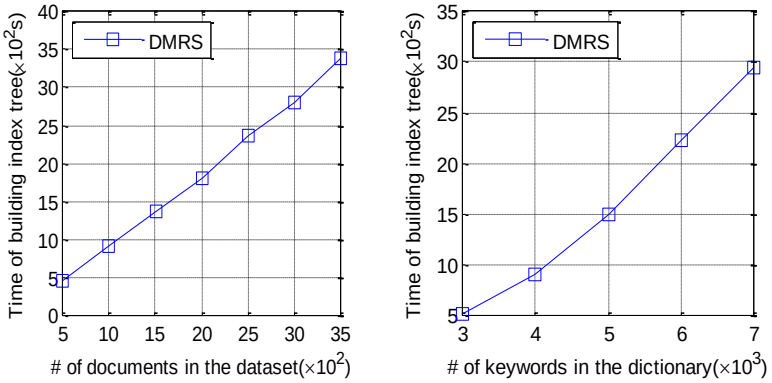
Security Analysis. To protect the security and privacy of index tree, we assume that there is a backup of unencrypted index data T stored at data owner side. Then, during the update operation, the encrypted subtree data P ¢is generated locally by data owners, and finally sent to the cloud server. As a result, the cloud server only receives a list of new nodes with encrypted information, which leaks no information about what have been updated.

4. Performance Analysis

In this section, we present a thorough performance evaluation of the proposed technique on a real-world dataset: the Request for Comments (RFC) [21]. The datasets used in the experiments are built from randomly chosen documents of different size. The entire secure search system is implemented using C++ language on a Windows Server with Intel Core(TM) Duo Processor 2. 93 GHz. The specific performance evaluation is presented as follows.

4.1. Index Construction

The process of index tree construction for dataset F includes two main steps: 1) build a MKRB index tree based on the whole dataset F ; 2) encrypt the index tree with splitting process and two multiplications of a $(m \times m)$ matrix. Since the index structure is constructed on the dataset F , so $O(n)$ nodes are visited during the procedure. Besides, for each node, index vector generation takes $O(m)$ time, vector splitting process takes $O(m)$ and two multiplications of a $(m \times m)$ matrix take $O(m^2)$, so, the whole time cost for index tree construction is $O(m^2 n)$ .Apparently, the time cost for building index tree mainly depends on the number of documents in dataset F and number of keywords in dictionary W . Figure 3(a) shows that the time cost for index tree construction is nearly linear with the size of dataset, given the same dictionary, *i.e.*, m is fixed. Figure 3(b) demonstrates that with the same dataset, the index construction time is proportional to the number of keywords in the dictionary. 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 red-black tree has space complexity $O(n)$ and every node stores a m -dimensional vector, it follows that the space complexity of the MKRB tree is $O(mn)$. As listed in Table 2, when the dataset is fixed (n=1000), the storage overhead of the index tree construction is determined by the size of the dictionary. Owing to the huge storage capacity and low storage overhead in the cloud, our schemes are practical and affordable.



(a) (b)

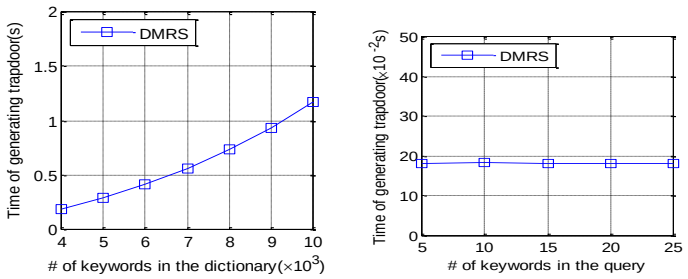
Figure 3. Time Cost for Index Tree Construction: (a) For Different Size of Dataset with the Same Dictionary, m=4000. (b) For Same Dataset with Different Size of Dictionary, n=1000

Table 2. Size of Index Tree

Size of dictionary	1000	2000	3000	4000	5000
Index tree size (MB)	73	146	220	293	367

4.2 Trapdoor Generation

Similar to the index construction, the generation of each trapdoor incurs a vector splitting process and two multiplications of a $(m \times m)$ matrix, so the time complexity is $O(m^2)$, *i.e.*, the time cost is determined by the number of keywords in the dictionary, as shown in Figure 4(a). Importantly, Figure 4(b) demonstrates that the number of query keywords has little influence on the overhead of trapdoor generation when the dictionary size is fixed.



(a) (b)

Figure 4. Time Cost for Trapdoor Generation: (a) For Same Query Keywords within Different Size of Dictionary, t=10. (b) For Different Numbers of Query Keywords with the Same Dictionary, m=4000

4.3. Search Efficiency

During the search process, if the similarity score at node u is larger than the minimum similarity score of the current selected top-k documents, the cloud server examines u 's children, 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 query as r, which is generally larger than k but far less than n. Since the maximum height of the red-black tree is maintained to be $\log n$, the search time without pruning is $O(rm \log n)$ (the complexity of similarity calculation is $O(m)$). However, different search paths may share same nodes during the search, and based on our "Greedy Depth-first Traverse Strategy", the search process terminates after the top-k documents have been selected. Thus, the number of actual accessed nodes is less than $r \log n$, which means the real search time is less than $O(rm \log n)$. Figure 5(a) shows the search time of DMRS with respect to the size of dataset, compared with MRSE in [13]. It is noteworthy that our proposed DMRS scheme is more efficient than MRSE which has linear search time, and with the increasing size of dataset, our scheme enjoys almost the same and nearly logarithmic search time. Figure 5(b) demonstrates that our search algorithm is still efficient when more relevant documents are retrieved.

4.4. Update Efficiency

According to analysis in Section 3, the update operation is merely executed on some specific nodes of a path from root to leaf in the index tree. Since the height of index tree is $\log n$, and encryption operation for index vector at each node takes $O(m^2)$ time, the update

complexity is $O(m^2 \log n)$. In Figure 6, we choose the deletion of a document as example. Figure 6(a) shows that when the size of dictionary is fixed, the deletion of a document takes nearly logarithmic time with the increasing size of dataset. And through Figure 6(b), we can see that the update time is proportional to the size of dictionary, while the dataset is fixed.

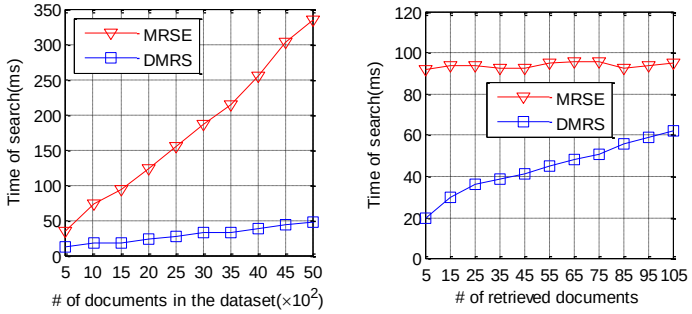


Figure 5. Search Efficiency with the Same 10 Inputted Keywords: (a) For Different Size of Dataset with the Same Dictionary, m=4000. (b) For Different Numbers of Retrieved Documents with the Same Dataset and Dictionary, n=1000, m=4000

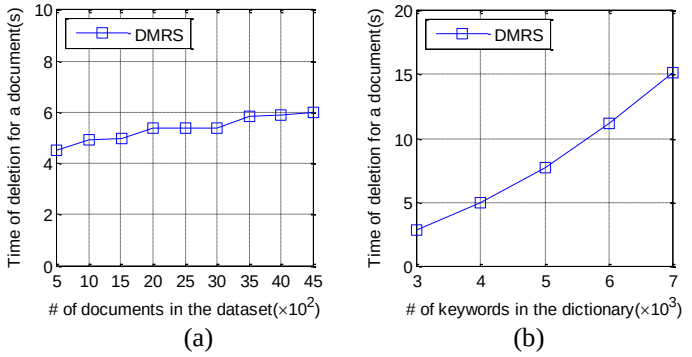


Figure 6. Time Cost for Deletion of a Document: (a) For the Different Size of Dataset with the Same Dictionary, m=4000. (b) For the Same Dataset with Different Size of Dictionary, n=1000

5. Conclusion

In this paper, we propose an efficient multi-keyword ranked search scheme over encrypted cloud data, which supports dynamic update operations. Among various multi-keyword semantics, we choose the popular one, *i.e.*, vector space model to present the relevance between documents and keywords. And cosine similarity measure is used to quantitatively evaluate the similarity between outsourced documents and query keywords, and furthermore achieve accurate ranked search results. With respect to search efficiency and update operations, we design a tree-based index and propose an efficient search algorithm. Moreover, in terms of privacy-preserving, we adopt a secure scheme in the known ciphertext threat model and successfully satisfy the privacy requirements. Eventually, experiments on the real-world dataset demonstrate the effectiveness and efficiency of our DMRS scheme. In

the future, we will concentrate on designing more efficient search algorithm and secure scheme in enhanced threat model.

Acknowledgements

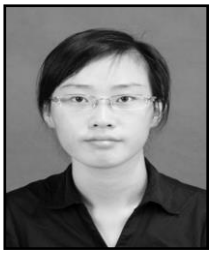
This work is supported by the NSFC (61232016, 61173141, 61173142, 61173136, 61103215, 61373132, 61373133), National Basic Research Program 973 (2011CB311808), 2011GK2009, GYHY201206033, 201301030, 2013DFG12860, SBC201310569 and PAPD fund.

References

[1] K. Ren, C. Wang and Q. Wang, "Security challenges for the public cloud", Internet Computing, IEEE, vol. 16, no. 1, **(2012)**, pp. 69-73.
[2] D. Boneh, E. Kushilevitz, R. Ostrovsky and W. E. Skeith III, "Public key encryption that allows PIR queries", Advances in Cryptology-CRYPTO 2007, Springer, **(2007)**, pp. 50-67.
[3] D. Boneh, G. Di Crescenzo, R. Ostrovsky and G. Persiano, "Public key encryption with keyword search", Advances in Cryptology-Eurocrypt 2004, **(2004)**, pp. 506-522.
[4] P. Van Liesdonk, S. Sedghi, J. Doumen, P. Hartel and W. Jonker, "Computationally efficient searchable symmetric encryption", Secure Data Management, Springer, **(2010)**, pp. 87-100.
[5] M. Bellare, A. Boldyreva and A. O'Neill, "Deterministic and efficiently searchable encryption", Advances in Cryptology-CRYPTO 2007, Springer, **(2007)**, pp. 535-552.
[6] D. X. Song, D. Wagner and A. Perrig, "Practical techniques for searches on encrypted data, Security and Privacy, 2000. S&P 2000", Proceedings. 2000 IEEE Symposium on, **(2000)**, pp. 44-55.
[7] Y.-C. Chang and M. Mitzenmacher, "Privacy preserving keyword searches on remote encrypted data", Applied Cryptography and Network Security, **(2005)**, pp. 442-455.
[8] R. Curtmola, J. Garay, S. Kamara and R. Ostrovsky, "Searchable symmetric encryption: improved definitions and efficient constructions", Proceedings of the 13th ACM conference on Computer and communications security, **(2006)**, pp. 79-88.
[9] S. Zerr, D. Olmedilla, W. Nejdl and W. Siberski, "Zerber+ r: Top-k retrieval from a confidential index", Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology, **(2009)**, pp. 439-449.
[10] W. Cong, C. Ning, R. Kui and L. Wenjing, "Enabling Secure and Efficient Ranked Keyword Search over Outsourced Cloud Data", Parallel and Distributed Systems, IEEE Transactions, vol. 23, no. 8, **(2012)**, pp. 1467-1479.
[11] D. Boneh and B. Waters, "Conjunctive, subset, and range queries on encrypted data", In Theory of cryptography, Springer, **(2007)**, pp. 535-554.
[12] J. Katz, A. Sahai and B. Waters, "Predicate encryption supporting disjunctions, polynomial equations, and inner products", Advances in Cryptology–EUROCRYPT 2008, Springer, **(2008)**, pp. 146-162.
[13] C. Ning, W. Cong, M. Li, R. Kui and L. Wenjing, "Privacy-preserving multi-keyword ranked search over encrypted cloud data, INFOCOM, 2011 Proceedings IEEE, **(2011)**, pp. 829-837.
[14] 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", Proceedings of the 8th ACM SIGSAC symposium on Information, computer and communications security, **(2013)**, pp. 71-82.
[15] C. D. Manning, P. Raghavan and H. Schütze, "Introduction to information retrieval", Cambridge University Press Cambridge, vol. 1, **(2008)**.
[16] S. Kamara, C. Papamanthou and T. Roeder, "Dynamic searchable symmetric encryption", Proceedings of the 2012 ACM conference on Computer and communications security, **(2012)**, pp. 965-976.
[17] S. Kamara and C. Papamanthou, "Parallel and dynamic searchable symmetric encryption", Financial Cryptography and Data Security, FC, **(2013)**
[18] S. Yu, C. Wang, K. Ren and W. Lou, "Achieving secure, scalable, and fine-grained data access control in cloud computing", INFOCOM, 2010 Proceedings IEEE, **(2010)**, pp. 1-9.
[19] C. E. Leiserson, R. L. Rivest, C. Stein and T. H. Cormen, "Introduction to algorithms", The MIT press, **(2001)**.
[20] W. K. Wong, D. W.-l. Cheung, B. Kao and N. Mamoulis, "Secure kNN computation on encrypted databases", Proceedings of the 2009 ACM SIGMOD International Conference on Management of data, **(2009)**, pp. 139-152.
[21] Request for Comments, http://www.rfc-editor.org/index.html, **(2011)**.

Authors

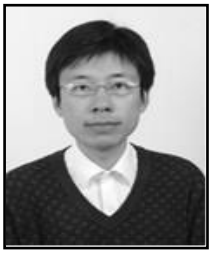
Xingming Sun received his BS in mathematics from Hunan Normal University, China, in 1984, MS in computing science from Dalian University of Science and Technology, China, in 1988, and PhD in computing science from Fudan University, China, in 2001. He is currently a professor in School of Computer & Software,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



Xinhui Wang received her BS 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.

Zhihua Xia received his BS 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.

Zhangjie Fu received his BS in education technology from Xinyang Normal University, China, in 2006; received his MS in education technology from the College of Physics and Microelectronics Science, Hunan University, China, in 2008; obtained his PhD in computer science from the College of Computer, Hunan University, China, in 2012. Currently, he works as an assistant professor in College of Computer and Software, Nanjing University of Information Science and Technology, China. His research interests include digital forensics, network and information security, copyright protection technology.



Tao Li received his MS degree in Computer Application from Nanjing University of Technology in 2004, and his PhD in Signal and Information Processing from Southeast University. He is currently working as a lecturer in School of Computer and Software, Nanjing University of Information Science and Technology. His research interests include wireless sensor network and embedded system.