International Journal of Multimedia and Ubiquitous Engineering
Vol.11, No.8 (2016), pp.107-120
http://dx.doi.org/10.14257/ijmue.2016.11.8.12

A Multi-keyword Ranked Search over Encrypted Cloud Data Supporting Semantic Extension

Zhihua Xia[1], Li Chen[1], Xingming Sun[1] and Jianxiao Liu[2]

[1]*Jiangsu Engineering Center of Network Monitoring, Jiangsu Collaborative Innovation Center on Atmospheric Environment and Equipment Technology, School of Computer & Software, Nanjing University of Information Science & Technology, Nanjing, 210044, China*
[2]*college of informatics, Huazhong Agricultural University, Hubei, China*
xia_zhihua@163.com, z_chenli@163.com, sunnudt@163.com and liujianxiao321@163.com

Abstract

With the emergence of cloud computing, many data owners outsource their local data to cloud server so as to enjoy high-quality data storage services. For the protection of data privacy, sensitive data has to be encrypted before outsourcing, which makes effective data utilization a challenging task. Although existing searchable encryption technologies enable data users to conduct secure search over encrypted data, the functionality of these schemes need to be further improved. In this paper, we construct a secure and efficient multi-keyword ranked search scheme which supports both the semantic extension search and the multi-keyword ranked search. The semantic extension is achieved through the mutual information statistical analysis of keywords. And the multi-keyword ranked search is achieved through a balanced binary tree whose nodes are the vectors of term frequency (TF) values. The splitting operation and secure transformation are utilized to encrypt the vectors of index and query. Note that, the encrypted vectors can be well used to calculate accurate relevance scores. Phantom terms are added to the index vector to blind the search results to resist statistical attacks. Due to the use of tree-based index structure, the proposed scheme can achieve the sub-linear search time. Finally, the experiments are conducted to demonstrate the efficiency of the proposed scheme.

Keywords*: multi-keyword ranked search; semantic search; mutual information; secure transformation; balanced binary tree*

1. Introduction

Nowadays, more and more attentions from industry and academia are paid to the cloud computing, which can provide huge resource of computing, storage and application [1]. Many people are motivated to outsource their local data (such as person health records, tax documents, financial transactions, and so on) to the cloud for its cost-efficiency and great flexibility [2]. However, the cloud server could only be semi-trusted, because they may learn the information of users' data and even leak users' data to some others. To protect data privacy, sensitive data has to be encrypted before outsourcing. However, this will cause a high cost in terms of data usability. For example, the existing techniques about keyword-based information retrieval, which are widely used on the plaintext data, cannot be directly applied on the encrypted data [3].

In order to address the above problem, searchable encryption (SE) techniques are proposed to provide secure search over encrypted data for users [4-8]. Song *et al.* [7] proposed the first symmetric searchable encryption (SSE) scheme. The search time of this scheme is linear to the size of the data collection. Goh *et al.* [8] proposed formal security

ISSN: 1975-0080 IJMUE

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 set.

Many inchoate methods only achieved exact single keyword search. To construct practical system, some researchers proposed the SE schemes to support multi-keyword ranked search [9-14]. This type of schemes allows user to input several query keywords to refine user's query. The search results are ranked according to some scoring criteria. This is a more practical type of technology. In order to deal with dynamic data collection, some researchers constructed dynamic schemes to support addition, deletion and modification document collection [15]. Specially, the dynamic search scheme has realized the multi-keyword ranked search functionality [14]. Considering that people may make spell errors when inputting query keywords, some researches proposed fuzzy keyword search schemes, which mainly employ a spell-check mechanism to support tolerance of minor typos [16-18]. These schemes mainly take the structure of terms into consideration and use edit distance to evaluate the similarity. They do not consider the terms semantically related to query keyword, thus many related files may be omitted.

Semantic search is a wide used technology to return more related results to user in plaintext search field [19-26]. In this paper, we propose a secure and efficient multi-keyword ranked search scheme, which takes both the semantic search and multi-keyword ranked search into consideration. The semantic extension is achieved through semantic relationship graph which is constructed by using the co-occurrence statistics of keywords. The multi-keyword ranked search is achieved through a balanced binary tree whose nodes are the vectors of term frequency (TF) values. Splitting operation, secure transformation and phantom terms are utilized to protect the data privacy. The proposed scheme achieves the sub-linear search time and can deal with the deletion and insertion of documents flexibly. In addition, the search efficiency of our scheme can be further increased by conducting parallel search on the tree index.

The reminder of the paper is organized as follows. In Section 2, we give a belief introduction to the system model, threat model, and design goals. Section 3 describes notations and preliminaries. Section 4 describes our scheme in detail. In Section 5, the search efficiency can be described. We conclude the paper in Section 6.

2. Problem Formulation

A. System Model

In the proposed SE scheme, the system model includes three entities: the data owner, the data user, and the cloud server.

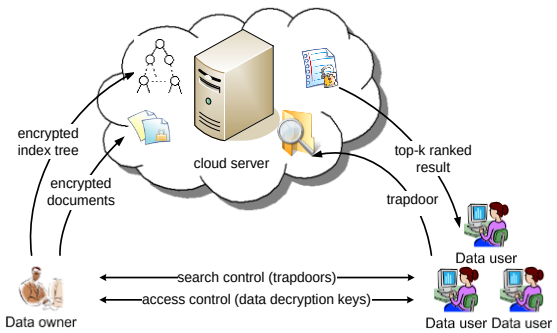


Figure 1. Architecture of Ranked Search Over Encrypted Cloud Data

Data owner has a collection of documents $D = \{d_1, d_2, ..., d_n\}$ and defines a dictionary of the keywords $W = \{w_1, w_2, ..., w_m\}$ from D . In the initialization of the system, the data owner constructs an encrypted searchable index tree I from the data collection D , and calculates

the inverted document frequency (*IDF*) values and semantic relationship graph (SRG) of the keywords. Next, all the files in D are encrypted to generate an encrypted data collection C. Finally, the data owner uploads both the encrypted index I and the collection C to cloud server, and distributes the secret keys of trapdoor generation and document decryption, the *IDF* values, and the SRG to the authorized data users.

Data users are the authorized ones to conduct search operation on cloud. To start a search, the data users take several keywords as input. First, the set of query keywords will be extended according to the SRG. Secondly, the weighted *IDF* values of keywords in the extended set are used to construct a query vector. Thirdly, phantom terms are added to the vector which is then encrypted by the splitting technique and secure transformation. Finally, the encrypted vector is used as the trapdoor which is submitted to the cloud to search the related documents. After cloud server returns top-k matching encrypted files, the data users can decrypt the files with the secret keys.

Cloud server stores the encrypted secure searchable index I and the collection of encrypted documents C. Upon receiving the trapdoor, the cloud server is responsible to execute search over the index and return the top-k ranked encrypted files to the data user. In addition, when the document collection has been updated by data owner, the cloud server also has to update the index and document collection stored in the server.

B. Threat Model

The cloud server is regarded as "honest-but-curious" in our system. That is to say, the cloud server honestly follows the designated protocols, while it is curious to analyze the stored information and learn additional information about the index and request. We consider two threat models with different attack capabilities as follows [9].

Known Ciphertext Model: In this model, the cloud server can only access the encrypted files, the secure index and the submitted trapdoor. Without the decryption keys, the cloud server cannot know the plaintext.

Known Background Model: In this stronger model, the cloud server knows more information such as term frequency, file frequency and co-occurrence statistic of the keywords [11]. With such statistical information, the cloud server could discern certain keywords by analyzing the list of search results or the encrypted index.

C. Design Goals

To enable effective and secure multi-keyword semantic ranked search over outsourced cloud data under the aforementioned model. The designed goals of our system are following:

Multi-keyword Semantic Ranked Search: The proposed scheme is designed to support semantic extension search and multi-keyword ranked search. Dynamic operation is supported to update the document collection.

Search Efficiency: The scheme aims to achieve sub-linear search efficiency by exploring a tree-based index and an efficient search algorithm. In addition, the parallel search can be conducted on the index to further improve the search efficiency.

Privacy-Preserving: Our scheme is designed to meet the privacy requirement and prevent the cloud server from learning additional information from index tree and trapdoor. Specifically, the privacy is preserved in following three aspects:

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, *IDF* values of query keywords, and the co-occurrence statistic of 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. Th*e cloud server could not identify the specific keyword in query, index or document collection by analyzing the statistical information like document/keyword frequency. Note that our proposed scheme is not designed to

protect access pattern, *i.e.*, the sequence of returned documents.

3. Notations and Preliminaries

A. Notations

- D –The plaintext collection, denoted as a set of n documents $D = \{d_1, d_2, ..., d_n\}$.
- C –The encrypted document collection stored in the cloud server, denoted as $C = \{c_1, c_2, ..., c_n\}$.
- W –The dictionary, the keyword set composing of m keyword, denoted as $W = \{w_1, w_2, ..., w_m\}$.
- W_q –The subset of W, denoting the set of keywords in a search request.
- W_s –The extensional semantic query keywords set for W_q.
- T – The unencrypted form of index tree for the whole document set D.
- I –The searchable encrypted tree index generated from T.
- $\mathbf{Q}$ –The query vector for keyword set W_s.
- T_{W_s} –The trapdoor for the search request. It is the encrypted form of $\mathbf{Q}$.
- $\mathbf{D}_v$ –The data vector for tree node v. Note that every node has a data vector $\mathbf{D}_v$ in the index tree. Specially, the node v can be either a leaf node or an internal node of the tree.
- $Score(\mathbf{D}_v, \mathbf{Q})$ –The function to calculate the similarity score for query vector $\mathbf{Q}$ and data vector $\mathbf{D}_v$ stored in node v.
- $k^{th}score$ –The smallest relevance score in current $RList$, which is initialized as 0.
- $hchild$ –The child node of a tree node with higher relevance score.
- $lchild$ –The child node of a tree node with lower relevance score.

B. Vector Space Model

Vector space model is the most popular similarity measure method in plaintext information retrieval, which supports both conjunctive search and disjunctive search [11]. Moreover, the vector space supports multi-keywords and non-binary presentation. Scoring is a natural way to rank relevant documents. In this paper, the documents are ranked according to the "*TF*×*IDF* rule". In the proposed scheme, each document is represented by a data vector $\mathbf{D}_v$, whose elements are normalized *TF* values of keywords in this document. And, a search request is expressed by a query vector $\mathbf{Q}$, whose elements are the normalized *IDF* value of query keywords in the document collection. Finally, the score of a data vector $\mathbf{D}_v$ on query $\mathbf{Q}$ is calculated by the inner product of the two vectors:

$$Score(\mathbf{D}_v, \mathbf{Q}) = \mathbf{D}_v \cdot \mathbf{Q} = \sum_{w_i \in W_s} TF_{v,w_i} \times IDF_{w_i} \quad , \tag{1}$$

Where TF_{v,w_i} denotes the normalized *TF* value of keyword w_i stored in index vector $\mathbf{D}_v$. If $\mathbf{D}_v$ is an internal node of the tree, TF_{v,w_i} is calculated from index vectors in the child nodes of *v*. If the index vector $\mathbf{D}_v$ is a leaf node, TF_{v,w_i} is calculated as:

$$TF_{v,w_i} = TF_{d,w_i}' \Big/ \sqrt{\sum_{w_i \in W} (TF_{d,w_i}')^2} \quad , \tag{2}$$

Where $TF_{d,w_i}' = 1 + \ln N_{d,w_i}$, and N_{d,w_i} denotes the number of keyword w_i in document d.

In the search vector $\mathbf{Q}$, IDF_{w_i} denotes the normalized *IDF* value of keyword w_i in document collection, and is calculated as:

$$IDF_{w_i} = IDF_{w_i}' \Big/ \sqrt{\sum_{w_i \in W_Q} (IDF_{w_i}')^2} \quad . \tag{3}$$

Specially, IDF_{w_i}' denotes the *IDF* value of keyword w_i in the document collection, and is calculated as:

$$IDF_{w_i}' = \ln\left(1 + N / N_{w_i}\right), \tag{4}$$

Where N_{w_i} denotes the number of documents that contain keyword w_i, and N denotes the total number of documents in the collection. To meet the need of data users, top-k relevant documents based on the scores are chose.

C. Keyword Balanced Binary tree

The keyword balanced binary (KBB) tree in our scheme is a dynamic data structure whose node stores a vector $\mathbf{D}$. The elements of vector $\mathbf{D}$ are the normalized *TF* values. Sometimes, we refer the vector $\mathbf{D}$ in the node v to $\mathbf{D}_v$ for simplicity. Formally, the node v in our KBB tree is defined as:

$$v = \langle ID, \mathbf{D}_v, P_l, P_r, FID \rangle, \tag{5}$$

Where ID denotes the identity of node v, P_l and P_r are respectively the pointers to the left and right child of node v. Here, if the node v is a leaf node of the tree, FID stores the identity of a document, and $\mathbf{D}_v$ denotes a vector consisting of the normalized *TF* values of the keywords in the document. If the node v is an internal node, FID is set to null, and $\mathbf{D}_v$ denotes a vector consisting of the *TF* values which is calculated as:

$$\mathbf{D}_v[i] = \max\{v.P_l \to \mathbf{D}_v[i], v.P_r \to \mathbf{D}_v[i]\}, i = 1, \ldots, m. \tag{6}$$

The detailed construction procedure of the tree-based index is presented in next section.

D. Secure k-NN computation

Secure k-nearest neighbor (*k*-NN) computation is such a method which can encrypt the vectors, and the encrypted vectors can still be used to calculate the accurate distances between each other [27-28]. During the encryption process of secure *k*-NN, a vector will be split into two vectors. The splitting technique is secure against known-plaintext attack[27]. Then, the vectors will be multiplied with the inventible matrices. Here, we need to calculate the dot product of two encrypted vectors, and denote the data vector as $\mathbf{D}_v$ and the query vector as $\mathbf{Q}$. The secret key is composed of one m bit vector as $\mathbf{S}$ and two $m \times m$ inventible matrices as $\{\mathbf{M}_1, \mathbf{M}_2\}$. First, $\mathbf{Q}$ is split into two random vectors as $\{\mathbf{Q}', \mathbf{Q}''\}$, and $\mathbf{D}_v$ is split into two random vectors as $\{\mathbf{D}_v', \mathbf{D}_v''\}$. The vector $\mathbf{S}$ is used as a splitting indicator. Specifically, if the j-*th* bit of $\mathbf{S}$ is 0, $\mathbf{D}_v'[j]$ and $\mathbf{D}_v''[j]$ are set as the same as $\mathbf{D}_v[j]$, while $\mathbf{Q}'[j]$ and $\mathbf{Q}''[j]$ are set to the random numbers so that their sum is equal to $\mathbf{Q}[j]$. If the j-*th* bit of $\mathbf{S}$ is 1, the splitting process is similar except that $\mathbf{D}_v$ and $\mathbf{Q}$ are switched. Next, the split data vector pair $\{\mathbf{D}_v', \mathbf{D}_v''\}$ is encrypted as $\{\mathbf{M}_1^T \cdot \mathbf{D}_v', \mathbf{M}_2^T \cdot \mathbf{D}_v''\}$, and the split query vector pair $\{\mathbf{Q}', \mathbf{Q}''\}$ is encrypted as $\{\mathbf{M}_1^{-1} \cdot \mathbf{Q}', \mathbf{M}_2^{-1} \cdot \mathbf{Q}''\}$. Finally, the formula to calculate the score is:

$$\begin{aligned} Score(\mathbf{D}_v, \mathbf{Q}) &= (\mathbf{M}_1^{\mathrm{T}} \mathbf{D}_v') \cdot (\mathbf{M}_1^{-1} \mathbf{Q}') + (\mathbf{M}_2^{\mathrm{T}} \mathbf{D}_v'') \cdot (\mathbf{M}_2^{-1} \mathbf{Q}'') \\ &= (\mathbf{M}_1^{\mathrm{T}} \mathbf{D}_v')^{\mathrm{T}} \mathbf{M}_1^{-1} \mathbf{Q}' + (\mathbf{M}_2^{\mathrm{T}} \mathbf{D}_v'')^{\mathrm{T}} \mathbf{M}_2^{-1} \mathbf{Q}'' \\ &= \mathbf{D}_v'^{\mathrm{T}} \mathbf{M}_1 \mathbf{M}_1^{-1} \mathbf{Q}' + \mathbf{D}_v''^{\mathrm{T}} \mathbf{M}_2 \mathbf{M}_2^{-1} \mathbf{Q}'' \\ &= \mathbf{D}_v' \cdot \mathbf{Q}' + \mathbf{D}_v'' \cdot \mathbf{Q}'' \\ &= \mathbf{D}_v \cdot \mathbf{Q}. \end{aligned} \tag{7}$$

After be split and transformed, neither query vector nor data vector can be guessed by analyzing their corresponding ciphertext. In addition, the Eq. (7) shows that we can

calculate the exact dot product of data vector $\mathbf{D}_v$ and the query vector $\mathbf{Q}$ by using their encrypted forms.

E.Semantic Relationship Graph

In this paper, we will construct the semantic relationship graph (SRG) by computing mutual information among keywords. Here, the statistical method proposed by Church and Hank [29] is adopted. For two keywords x and y, their mutual information $I(x,y)$ is defined as:

$$I(x,y) = \log_2 \frac{p(x,y)}{p(x)p(y)}, \tag{8}$$

Where $p(x)$ and $p(y)$ are the ratios of documents that contain the keyword x and y respectively, and $p(x,y)$ is the ratio of documents that contain both the keyword x and the keyword y. In this paper, the mutual information is normalized into as interval $(0,1]$ as:

$$I(x,y) \leftarrow \frac{I(x,y)}{I_{\max} \cdot \boldsymbol{\alpha}}, \tag{9}$$

Where $I_{\max}$ denotes the max value in the all mutual information $I(x,y)$, and α is a factor used to scale $I(x,y)$ so as to adjust the importance of the extended keywords in the search.

Figure. 2 shows a small scale semantic relationship graph, whose nodes denote the keywords and edges are the normalized mutual information between the corresponding two keywords. Here, the factor α is set to 1. Church and Hank [29] suggested that there is a genuine association between x and y when $I(x,y) > 0$. Thus, we only add an edge between two nodes when the mutual information between them is greater than 0.

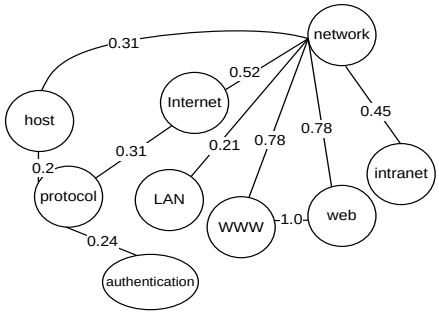


Figure 2. An Example of Semantic Relationship Graph

4. The Proposed Scheme

In this paper, we exploit the KBB tree together with the semantic relationship graph to achieve semantic and privacy-assured multi-keyword ranked search over encrypted cloud data. First, the construction of unencrypted KBB tree is introduced. Secondly, the search process on the unencrypted index is presented. Thirdly, we introduce construction for multi-keyword ranked retrieval search scheme which add the encryption to construction of the index and query trapdoor.

A. The Construction of Unencrypted Index Tree

In the Section 3, the construction of the nodes of the KBB index tree is briefly introduced. In the process of index construction, we first generate a tree node for each document in the collection. These nodes are used as the leaf nodes of the index tree. The values of leaf node are the *TF* values of keywords in the corresponding document. Secondly, the internal nodes of the tree are generated based on these leaf nodes according

to the formula (6). The construction process of the index tree is presented in Algorithm 1; and an example of unencrypted index tree is shown in Figure. 5. Following are the two symbols in Algorithm 1.

- $CurrentNodeSet$ – 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 $2h(h \in Z^+)$, else the cardinality is denoted as $(2h+1)$.
- $TempNodeSet$ – The set of the newly generated nodes.

Algorithm 1 *Index Tree Construction*

$BuildIndexTree(\mathrm{D}) \rightarrow \mathrm{T}$:

procedure

1. **Initialization:** Input the document collection $\mathrm{D} = \{d_1, d_2, ..., d_n\}$ with the identifiers $\mathrm{FID} = \{FID \mid FID = 1, 2, ..., n\}$; Define a function $GenID()$ to generate a unique identity for each tree node.
2. **Pad data to all nodes of index tree:**

Define the data structure of each node as $v = \langle ID, \mathbf{D}_v, P_l, P_r, FID \rangle$ in the index tree;

for(each document d_{FID} **in** D **)**/*set the data structure of leaf nodes, as Figure.5*/

Set $v.ID = GenID()$, $v.P_l = null, v.P_r = null, v.FID = FID$,

$\mathbf{D}[i] = TF_{d_{FID}, w_i}$, for $i = (1, 2, ..., m)$;

Insert v to $CurrentNodeSet$;

end for

for (the number of nodes in $CurrentNodeSet$ is larger than 1) **do**

if (the number of nodes in $CurrentNodeSet$ is even, *i.e.* $2h$) **then**

for (each pair of nodes v' and v'' in $CurrentNodeSet$) **do**

Generate a parent node v for v' and v'';

$v.ID = GenID()$, and $v.P_l = v', v.P_r = v'', v.FID = 0, D_v[i] = \max\{v'.D_v[i], v''.D_r[i]\}$ for each $i = 1, ..., m$;

Insert v to $TempNodeSet$;

end for

else /* the number of nodes in $CurrentNodeSet$ is odd, *i.e.* $(2h+1)$ */

for (each pair of nodes v' and v'' of the former $(2h-2)$ nodes in $CurrentNodeSet$) **do**

Generate a parent node v for v' and v'';

Insert v to $TempNodeSet$;

end for

Create a parent node $v1$ for the $(2h-1)$-*th* and $2h$-*th* node, and then create a parent node v for $v1$ and the $(2h+1)$-*th* node;

Insert v to $TempNodeSet$;

end if /***if** (the number of nodes in $CurrentNodeSet$ is even, *i.e.* $2h$)*/

Replace $CurrentNodeSet$ with $TempNodeSet$ and then clear $TempNodeSet$;

end for /* **for** (the number of nodes in $CurrentNodeSet$ is larger than 1)***/**

return the only node in $CurrentNodeSet$, namely, the root of index tree T;

In the index, if $\mathbf{D}_u[i] \neq 0$ for an internal node v, there is at least one path from the

node v to some leaf, which indicates a document d containing the keyword w_i. In addition, $\mathbf{D}_u[i]$ 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.

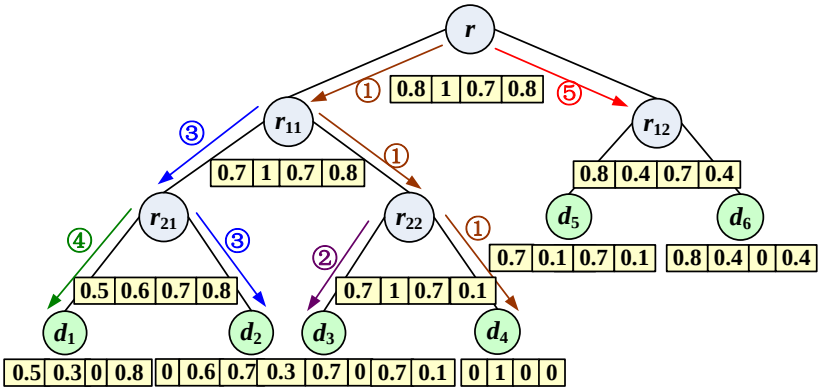


Figure 5. An Example of the Tree-Based Index

An example of the tree-based index is illustrated in Figure 5 where the document collection $\mathrm{D}=\{d_i \mid i=1,...6\}$ 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 $\mathbf{Q}$ 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 d4 through r11 and r22. The relevance score of d4 to the query is 0.92. After that, the leaf nodes d3 and d2 are successively reached with the relevance scores 0.038 and 0.67. Next, the leaf noded1 is reached and replace d3 in $RList$. Finally, the algorithm will try to search subtree rooted by r12, and find that there will be no reasonable results in this subtree because the relevance score of r12 is 0.52, which is smaller than the smallest relevance score in $RList$. Finally, the search results include d4, d3, and d1.

B. Search Process on Unencrypted Index Tree

The search process of our scheme is a recursive procedure upon the tree, named as 'Greedy Depth-first Search (*GDFS*)' algorithm. Since the internal node stores the bigger normalized *TF* values between its child nodes, it is easy to determine which sub-tree can be firstly searched in the light of the query vector. The search starts with the root node. First of all, the search compares the inner product of the two child nodes of root node, and then decides which sub-tree is preferentially searched. After that, this procedure is executed recursively until the top-k objects is selected.

We construct a result list denoted as $RList$, whose element is defined as $\langle Score, FID \rangle$. Specifically, the $Score$ represents the relevance of the document d_{FID} to the query, and 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 $Score$, and will be updated timely during the search process. Here, $k^{th}score$ denotes the smallest score in the $RList$. Following are the detail of *GDFS* algorithm in Algorithm 2.

Algorithm 2 *GDFS* Algorithm on index tree T

$RList \leftarrow GDFS(\text{IndexTreeNode } v)$:
if (the node v is not a leaf node) **then**
if ($Score(\mathbf{D}_v, \mathbf{Q}) > k^{th}score$) **then**
$GDFS(v.hchild)$;
$GDFS(v.lchild)$;
else return;
end if
else if ($Score(\mathbf{D}_v, \mathbf{Q}) > k^{th}score$) **then**
Delete the element with the smallest relevance score from $RList$;
Insert a new element $\langle Score(\mathbf{D}_v, \mathbf{Q}), v.FID \rangle$ and sort all the elements of $RList$;
end if
return;
end if

C. The Construction for Scheme

In this subsection, we construct the multi-keyword ranked search scheme by integrating splitting operation and secure transformation to the unencrypted index. The scheme includes two phases: the construction of encrypted index and the search process on index tree.

1) The Construction of Encrypted Index

- $Setup(\lambda) \rightarrow \mathrm{SK}$: The setup algorithm takes λ as the security parameter. The data owner generates a random $(m+m')$-bit vector $\mathbf{S}$ as a spiting indicator and generates two $(m+m') \times (m+m')$ invertible matrices $\{\mathbf{M}_1, \mathbf{M}_2\}$ as encryption matrix. Specifically, m is the size of dictionary W, and m' is the number of dummy terms. The data owner outputs a symmetric key as $\mathrm{SK} = \{\mathbf{S}, \mathbf{M}_1, \mathbf{M}_2\}$.
- $GenSecureIndex(\mathrm{D}, \mathrm{SK}) \rightarrow \mathrm{I}$: First, the algorithm $BuildIndexTree(\mathrm{D}) \rightarrow \mathrm{T}$ is called to generate the unencrypted index tree T. Secondly, before encrypting the index vector $\mathbf{D}_v$ of T, the data vector $\mathbf{D}_v$ is extended from m-dimensions to $(m+m')$-dimensions. Each extended element $\mathbf{D}_v[m+j]$, for $j = 1, ..., m'$ is set as a random number ε_j. Thirdly, the data vectors are split according to the indicate vector $\mathbf{S}$ after applying dimension-extending. The index vector $\mathbf{D}_v$ for each node v is split two random vectors $\{\mathbf{D}'_v, \mathbf{D}''_v\}$. Rules are as follows: if $\mathbf{S}[i] = 0$, $\mathbf{D}'_v[i]$ and $\mathbf{D}''_v[i]$ is set equal to $\mathbf{D}_v[i]$. If $\mathbf{S}[i] = 1$, $\mathbf{D}'_v[i]$ and $\mathbf{D}''_v[i]$ is set as two random values whose sum equals to $\mathbf{D}_v[i]$. Fourthly, the split data vectors $\{\mathbf{D}'_v, \mathbf{D}''_v\}$ are encrypted by the invertible matrices $\{\mathbf{M}_1, \mathbf{M}_2\}$. Finally, the encrypted index tree I is built where the node v stores two encrypted index vectors $\mathbf{I}_v = \{\mathbf{M}_1^T \mathbf{D}'_v, \mathbf{M}_2^T \mathbf{D}''_v\}$. The data owner outsources the encrypted searchable index tree to the cloud server until the whole vectors stored in the index tree nodes are encrypted. In addition, the encrypted document collection is outsourced.

The *IDF* values and the SRG also need to be calculated by data owner and distributed to authorized data users. The *IDF* values of the keywords in the dictionary are calculated according to the Formula (4). And SRG is constructed as it is introduced in Section 3. In

addition, the documents are encrypted by symmetric key cryptography, and the encrypted documents are uploaded to the cloud server.

2) The Search Process on Index Tree

- $GenTrapdoor(\tilde{W}, SK) \rightarrow T_{\tilde{W}_s}$: With interest keyword set $\tilde{W}$, this algorithm semantically extends the query keywords based on the SRG to gain the extensional query keywords set $\tilde{W}_s$. In order to tell the important degree between the original keywords and the extensional keywords, it has to take the edge's weight of SRG into consideration. First, $\mathbf{Q}[i]$ will store a value associated with w_i. The value is calculated including three aspects :1) we set the semantic weight values of original keywords as 1 and get the edge's weight values between the original keywords and the extensional keywords from SRG as the corresponding semantic weight values. However, when an extensional keyword is semantically related to several original keywords, we select the maximum value of edge's weight values between the original keywords and the extensional keyword from SRG as its semantic weight value; 2) If the keyword w_i is contained in $\tilde{W}$, $\mathbf{Q}[i]$ stores the *IDF* value of w_i. If the keyword w_i is an extensional keyword, $\mathbf{Q}[i]$ stores the product of *IDF* value of w_i and the semantic weight edge's value between w_i and the original keyword. If the keyword w_i is an extensional keyword and has more than two semantic related original keywords, $\mathbf{Q}[i]$ stores the product of *IDF* value of w_i and the maximum edge's weight value. If the keyword w_i is not contained in $\tilde{W}_s$, $\mathbf{Q}[i]$ is set as 0; 3) Taking advantage of the formula (3), the query vector $\mathbf{Q}$ need to be normalized. After these series of process, the query vector $\mathbf{Q}$ stores the normalized value. Before encrypting the query vector $\mathbf{Q}$, we extend the dimension of query vector $\mathbf{Q}$ from m to $(m+m')$, secondly. Thirdly, we choose randomly m'' $(m'' < m')$ elements from m' elements set their values as 1, the rest are set as 0. Fourthly, query vector $\mathbf{Q}$ is split two random vectors $\{\mathbf{Q}', \mathbf{Q}''\}$. On the contrary, if $\mathbf{S}[i] = 0$, $\mathbf{Q}'[i]$ and $\mathbf{Q}''[i]$ are set as two random values whose sum equals to $\mathbf{Q}[i]$. If $\mathbf{S}[i] = 1$, $\mathbf{Q}'[i]$ and $\mathbf{Q}''[i]$ are set equal to $\mathbf{Q}[i]$. Finally, the algorithm returns the trapdoor $T_{\tilde{W}_s} = \{\mathbf{M}_1^{-1}\mathbf{Q}', \mathbf{M}_2^{-1}\mathbf{Q}''\}$. And it needs to submit the trapdoor to the cloud server.
- $Search(I, T_{\tilde{W}_s}, K) \rightarrow R$: With the trapdoor and the parameter k, the cloud server executes the algorithm.2 computes the relevance score of node v in the index tree I to the query. The computation is described as in formula (10), where $r \in \{j \mid \mathbf{Q}[m+j] = 1\}$. According to the algorithm.2, the cloud server returns top-k identities of relevant documents to the data user.

$$\begin{aligned}
&\mathbf{I}_v \cdot T_{\tilde{W}_s} \\
&= (\mathbf{M}_1^T \mathbf{D}_v') \cdot (\mathbf{M}_1^{-1}\mathbf{Q}') + (\mathbf{M}_2^T \mathbf{D}_v'') \cdot (\mathbf{M}_2^{-1}\mathbf{Q}'') \\
&= \mathbf{D}_v'^T \mathbf{M}_1 \mathbf{M}_1^{-1}\mathbf{Q}' + \mathbf{D}_v''^T \mathbf{M}_2 \mathbf{M}_2^{-1}\mathbf{Q}'' \\
&= \mathbf{D}_v' \cdot \mathbf{Q}' + \mathbf{D}_v'' \cdot \mathbf{Q}'' \\
&= \mathbf{D}_v \cdot \mathbf{Q} + \sum \varepsilon_r \\
&= Score(\mathbf{D}_v, \mathbf{Q}) + \sum \varepsilon_r
\end{aligned} \tag{10}$$

5. Performance

In this section, we show experimental evaluations of the proposed technique on a real dataset: the RFC dataset [30]. 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. The search efficiency of the proposed scheme is compared with the method which is presented in [10].

1) Search efficiency

During the search process, if the relevance score at node v is larger than the minimum relevance score in result list $RList$, the cloud server examines the children of the node; else it returns. Therefore, 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. As a balanced binary tree, the height of the balanced binary tree is maintained to be $\log n$, and the complexity of relevance score calculation is $O(m)$. Thus, the time complexity of search is $O(\theta m \log n)$. Note that the real search time is less than $O(\theta m \log n)$. The reasons includes three aspects: 1) according to our search algorithm, many leaf nodes that contain the queried keywords are not visited, 2) the accessing paths of some different leaf nodes share the mutual traversed parts, 3) the parallel execution of search process can improve the search efficiency .

The search process executed at the cloud server involves searching and computing the similarity score. The search algorithm terminates after the top-k files have been returned. 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 in our experiments. Figure 6(a) shows that when the size of dictionary and document collection are fixed, the search time vary with the parameter k. In Figure 6(b), we compare the search efficiency of our scheme with the recent work by Sun *et al.*[10]. For Sun *et al.*'s method, we choose 4000 keywords and divide them in 50 levels. Consequently, each level contains 80 keywords. The experimental results show the higher level the query keywords reside, the higher the search efficiency is. We choose ten keywords from the 1st level (the highest level, the optimal case) and another ten keywords from the 26th level (the middle level, the average normal case) respectively for search efficiency comparison. Figure.6 (b) shows that if the query keywords come from the middle level of index, the search efficiency of our scheme is more efficient than Sun *et al.*'s work. If the query keywords are chosen from 1st level, our scheme obtains almost the same efficiency as the Sun *et al.*'s scheme when we start 4 threads.

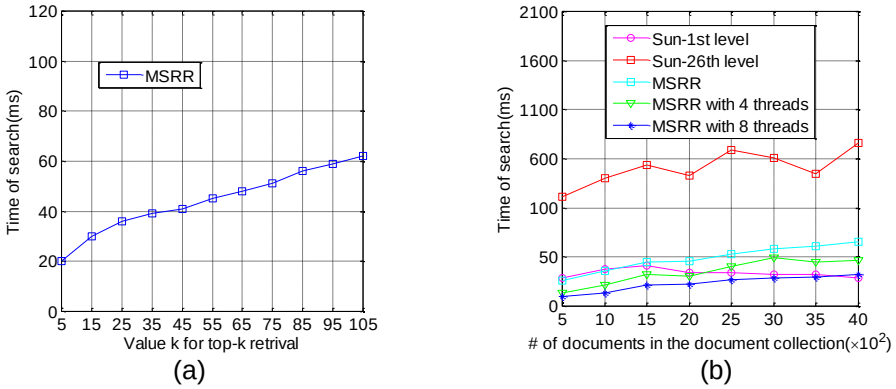


Figure 6. The Efficiency of a Search: (a) The Time Cost for Top-k Retrieval with the Same Dictionary and Document Collection, $n=1000, m=4000$, (b) the Time Cost for Different Document Collection with Different Methods

6. Conclusion

In this paper, we solve the problem of multi-keyword ranked search over the encrypted cloud data, and establish a variety of privacy requirement. The proposed scheme can return not only the exact matched files, but also the files including the terms semantically related to the query keywords. We construct the index tree whose data vectors store the *TF* values to improve the search efficiency effectively. When inputting the query keywords, keywords first are extended according to the semantic relationship graph. Specially, the query vector is expressed as the *IDF* values of query keywords and extensional keywords. After that, the trapdoor is uploaded to the cloud server. The cloud server explores the "inner product similarity" to quantitatively evaluate similarity measure to capture the relevance of outsourced documents to the query request. According to the search algorithm, the semantic related top-k files are returned. Taking security and privacy into consideration, a secure splitting *k-NN* technique is employed to encrypt the index and query vector, so that we can obtain the accurate ranked results and perform well protection on the confidence of the data.

As our future work, we will concentrate on the encrypted data of semantic keyword search in order to confront with the more sophisticated search.

Acknowledgments

This work is supported by the NSFC (61173141, 61232016, 61572258, 61502242, U1405254, 61173136, 61373133), 201301030, 2013DFG12860, BC2013012, Fund of Jiangsu Engineering Center of Network Monitoring (KJR1308, KJR1402), Fund of MOE Internet Innovation Platform (KJRP1403), CICAEET, and PAPD fund. Zhihua Xia is supported by Jiangsu Government Scholarship for Overseas Studies (JS-2014-332).

References

[1] L. M. Vaquero, L. R. Merino, J. Caceres and M. Lindner, "A break in the clouds: towards a cloud definition", ACM SIGCOMM Computer Communication Review, vol. 39, **(2008)**, pp. 50-55.
[2] S. Kamara and K. Lauter, "Cryptographic cloud storage", Financial Cryptography and Data Security, pp. **(2010)**, pp. 136-149.
[3] K. Ren, C. Wang and Q. Wang, "Security challenges for the public cloud", IEEE Internet Computing, vol. 16, **(2012)**, pp. 69-73.
[4] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky, "Searchable symmetric encryption: improved definitions and efficient constructions", presented at the Proceedings of the 13th ACM conference on Computer and communications security, Alexandria, VA, USA, **(2006)**.
[5] M. Bellare, A. Boldyreva, and A. O'Neill, "Deterministic and Efficiently Searchable Encryption", Advances in Cryptology - CRYPTO 2007, vol. 4622, **(2007)**, pp. 535-552.
[6] 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. vol. 8042, R. Canetti and J. Garay, Eds., ed: Springer Berlin Heidelberg, **(2013)**, pp. 353-373.
[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, **(2000)**, pp. 44-55.
[8] E.-J. Goh, "Secure Indexes", IACR Cryptology ePrint Archive, vol. 2003, **(2003)**, p. 216.
[9] N. Cao, C. Wang, M. Li, K. Ren, and W. Lou, "Privacy-preserving multi-keyword ranked search over encrypted cloud data", Parallel and Distributed Systems, IEEE Transactions on, vol. 25, **(2014)**, pp. 222-233.
[10] W. Sun, B. Wang, N. Cao, M. Li, W. Lou and Y. T. Hou, "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, **(2013)**, pp. 71-82.
[11] J. Yu, P. Lu, Y. Zhu, G. Xue, and M. Li, "Towards Secure Multi-Keyword Top-k Retrieval over Encrypted Cloud Data", IEEE transactions on dependable and secure computing, **(2013)**, p. 1.
[12] J. Xu, W. Zhang, C. Yang, J. Xu, and N. Yu, "Two-Step-Ranking Secure Multi-Keyword Search over Encrypted Cloud Data", in Cloud and Service Computing (CSC), 2012 International Conference on, **(2012)**, pp. 124-130.
[13] F. Zhangjie, S. Xingming, L. Qi, Z. Lu, and S. Jiangang, "Achieving Efficient Cloud Search Services: Multi-keyword Ranked Search over Encrypted Cloud Data Supporting Parallel Computing", IEICE Transactions on Communications, vol. 98, **(2015)**, pp. 190-200.

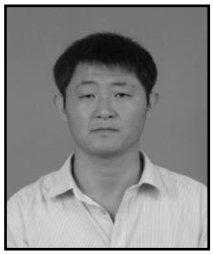
[14] Z. Xia, X. Wang, X. Sun, and Q. Wang, "A Secure and Dynamic Multi-keyword Ranked Search Scheme over Encrypted Cloud Data", Parallel and Distributed Systems, IEEE Transactions on, **(2015)**, pp. 1-1.
[15] S. Kamara and C. Papamanthou, "Parallel and dynamic searchable symmetric encryption", in Financial Cryptography and Data Security, ed: Springer, **(2013)**, pp. 258-274.
[16] M. Chuah and W. Hu, "Privacy-aware bedtree based solution for fuzzy multi-keyword search over encrypted data", in Distributed Computing Systems Workshops (ICDCSW), 2011 31st International Conference on, **(2011)**, pp. 273-281.
[17] W. Zhou, L. Liu, H. Jing, C. Zhang, S. Yao, and S. Wang, "K-Gram Based Fuzzy Keyword Search over Encrypted Cloud Computing", **(2013)**.
[18] 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, **(2010)**, pp. 1-5.
[19] F. Lamberti, A. Sanna, and C. Demartini, "A relation-based page rank algorithm for semantic web search engines", Knowledge and Data Engineering, IEEE Transactions on, vol. 21, **(2009)**, pp. 123-136.
[20] J. Hendler, "Web 3. 0: The Dawn of Semantic Search", Computer, vol. 43, **(2010)**, pp. 77-80.
[21] J. Wei, S. Bressan, and B. C. Ooi, "Mining term association rules for automatic global query expansion: methodology and preliminary results", in Web Information Systems Engineering, 2000. Proceedings of the First International Conference on, **(2000)**, pp. 366-373.
[22] D. Pal, M. Mitra, and K. Datta, "Query expansion using term distribution and term association", arXiv preprint arXiv:1303.0667, **(2013)**.
[23] L.-F. Lai, C.-C. Wu, P.-Y. Lin, and L.-T. Huang, "Developing a fuzzy search engine based on fuzzy ontology and semantic search", in Fuzzy Systems (FUZZ), 2011 IEEE International Conference on, **(2011)**, pp. 2684-2689.
[24] B. M. Fonseca, P. B. Golgher, E. S. De Moura, B. Pôssas, and N. Ziviani, "Discovering search engine related queries using association rules", Journal of Web Engineering, vol. 2, **(2003)**, pp. 215-227.
[25] M. Song, I.-Y. Song, X. Hu, and R. Allen, "Semantic query expansion combining association rules with ontologies and information retrieval techniques", in Data Warehousing and Knowledge Discovery, ed: Springer, **(2005)**, pp. 326-335.
[26] M. Song, I.-Y. Song, X. Hu, and R. B. Allen, "Integration of association rules and ontologies for semantic query expansion", Data & Knowledge Engineering, vol. 63, **(2007)**, pp. 63-75.
[27] 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, **(2009)**, pp. 139-152.
[28] J. Li, X. Li, B. Yang, and X. Sun, "Segmentation-based Image Copy-move Forgery Detection Scheme", Information Forensics and Security, IEEE Transactions on, vol. 10, **(2015)**, pp. 507-518.
[29] K. W. Church and P. Hanks, "Word association norms, mutual information, and lexicography", Computational linguistics, vol. 16, **(1990)**, pp. 22-29.
[30] B. Gu, Victor S. Sheng, K. Y. Tay, W. Romano, and S. Li, "Incremental Support Vector Learning for Ordinal Regression", IEEE Transactions on Neural Networks and Learning Systems, vol. 26, no. 7, pp. 1403-1416.
[31] Z. Pan, Y. Zhang, and S. Kwong, "Efficient motion and disparity estimation optimization for low complexity multiview video coding", IEEE Transactions on Broadcasting, vol. 66, no. 2, pp. 166-176.
[32] "Request for Comments. Available", http://www.rfc-editor.org/index.html, **(2011)**.

Authors

Zhihua Xia, received the BS degree in Hunan City University, China and PhD degree in computer science and technology from Hunan University, China, in 2006 and 2011, respectively. He works as an associate professor in School of Computer & Software, Nanjing University of Information Science & Technology. His research interests include digital forensic and encrypted image processing. He is a member of the IEEE.

Li Chen, She is currently pursuing her MS in computer science and technology at the College of Computer & Software, in Nanjing University of Information Science & Technology, China. Her research interests include keyword search over encrypted cloud data.

Xingming Sun, received his PhD in computing science from Fudan University, China, in 2001. He is currently a professor in Nanjing University of Information Science & Technology, China. His research interests include network and information security, digital watermarking.



Jianxiao Liu, is a lecture in college of informatics of Huazhong Agricultural University. His current research interest is service computing and machine learning.