A Verifiable and Dynamic Multi-keyword Ranked Search Scheme over Encrypted Cloud Data with Accuracy Improvement

Qi Zhang[1(✉)], Shaojing Fu[1,2,3], Nan Jia[1], and Ming Xu[1]

[1] College of Computer, National University of Defense Technology, Changsha, China
zqi6@outlook.com
[2] State Key Laboratory of Cryptology, Beijing, China
[3] Science and Technology on Information Assurance Laboratory, Beijing, China

Abstract. With the widely application of cloud computing, more and more data owners prefer to outsource their data on the remote cloud servers to reduce the overhead. Searchable encryption is proposed in an urgently need for searching on the encrypted data. In this paper, we present a tree-based privacy-preserving and efficient multi-keyword ranked search scheme supporting verification and dynamic update. Considering the effect of the keywords location on the weight in most documents, the traditional $TF \times IDF$ algorithm can be optimized with location information to get more accurate similarity score. To improve the efficiency, we combine the vector space model and binary tree to construct a tree-based index structure. And the index tree is encrypted by secure kNN computation. Finally, We analyze the security against two threat model, and implement the experiment on the real paper set to evaluate the performance.

Keywords: Keyword location · Searchable symmetric encryption
Tree-based index · Verification · Dynamic update

1 Introduction

In recent years, cloud computing has sprung up in various fields due to its unique advantages. More and more data owners choose to outsource their large amounts of local data to remote cloud servers to reduce their storage and computation overhead.

Despite those benefits, data stored on the cloud servers, especially the sensitive information, faces serious security risks and privacy challenges since cloud servers are honest-but-curious. A general way to reduce this leakage is encrypting the data before outsourcing. However, encryption will bring other difficulties. For example, if we want to search an exact document on the server, we have to download all the encrypted data and decrypt it locally which lead to large storage and computation cost. Therefore, searching on encrypted data becomes a valuable research issue.

R. Beyah et al. (Eds.): SecureComm 2018, LNICST 254, pp. 588–604, 2018.
https://doi.org/10.1007/978-3-030-01701-9_32

Searchable encryption is proposed to settle these problems since it can guarantee the security and usability of data. These years, lots of works has been proposed on this field, such as single keyword search, multi-keyword search, ranked search. Furthermore, verification and dynamic update are added to fulfill the functionality.

In this paper, we present a tree-based secure and efficient multi-keyword ranked search scheme supporting verification and dynamic update. Our paper takes the keyword location into account. In plaintext area, the information retrieval mechanism is first returning all the results whose title contains the keywords and later returning the results whose body contains them. That is to say the keyword location will greatly influence the similarity of the documents with query request. However, in ciphertext, traditional TF value only consider the number of the keywords in a document. Introducing the location information in the TF value will counts. Moreover, the length of the paper also have impact on the similarity. We construct a tree-based index structure to improve the efficiency of search. Moreover, the verification and dynamic update function are also designed based on the tree-base index structure. We choose to implement the experiment on the data set of paper for its typical fixed format which composed of title, abstract, body, conclusion and preferences so that we can easily assign the different significance to the keyword in different part. Our Contribution can be summarized as follows:

(1) We present a tree-based secure and efficient multi-keyword ranked search scheme supporting verification and dynamic update.
(2) We introduce the keyword location and length of document to optimize the $TF \times IDF$ algorithm to improve the accuracy.
(3) We analyze the security against two threat model, and implement the experiment on the real paper set to evaluate the performance.

The rest of this paper is organized as follows. Section 2 gives the related word on searchable encryption. Section 3 gives a brief introduction of the scheme. Section 4 describes the scheme in detail. Section 5 presents the performance analysis. Finally, we conclude in Sect. 6.

2 Related Work

Song et al. [15] first proposed a solution for searching single keyword on encrypted data with sequential scan which was provably secure but in high cost. Boneh [1] first proposed public key searchable encryption scheme. Based on these scheme, a great deal of improvement had been produced.

Single Keyword Search. Goh [8] defined a secure index using a bloom filter and pseudo-random functions which will introduce false positive results. Chang and Mitzenmacher [4] developed two index schemes using dictionaries. Li et al. [14] developed a single keyword search scheme to support fuzzy search.

Multi-keyword Search. Lots of research [2,9,19] achieve the conjunctive multi-keyword search. Boneh et al. [2] proposed a public-key scheme and supported conjunctive and disconjunctive search like subset and range query. Wang et al. [19] designed a inverted index based public-key searchable encryption scheme. They used private set intersection to support conjunctive multi-keyword search. Wang et al. [21] utilized the bloom filter with LSH and construct two schemes using homomorphic encryption and pseudorandom padding to deal with high-dimensional feature-rich data.

Rank Search. Rank search is proposed to deal with the drawbacks of boolean search. Wang et al. [20] used inverted index and $TF \times IDF$ to construct a order-preserving symmetric encryption. However this scheme only support single keyword search. Cao et al. [3] first proposed a basic multi-keyword ranked search scheme (MRSE) using secure inner product computation which had low overhead on computation and communication. However, this scheme ignored the different importance of the keywords. Fu et al. [7] proposed a scheme supporting both fuzzy and rank search by processing stemming algorithm, LSH and bloom filter. Sun et al. [17] developed the scheme by MDB-tree index structure. Chen et al. [5] used k-means algorithm to construct the hierarchical cluster index tree. Xia et al. [22] designed a special KBB index tree to provide efficient multi-keyword ranked search which we refer to in this paper.

Dynamic Search. In practice, the data on the server could not be immutable. Therefore, update should be considered. Goh [8] realized the update based on the bloom filter. Kamara et al. [11] constructed a new dynamic encrypted index to give a dynamic SSE scheme. Later, they improved the scheme by KBB tree in [10]. Wang et al. [21] provide efficient index dynamic over homomorphic encryption and pseudorandom padding. Wan and Deng [18] applied update based on bilinear-map accumulation tree. Lai and Chow [12] developed a dynamic symmetric structured encryption scheme with random binary tree.

Verifiable Search. Wan and Deng [18] gave a solution to apply verification based on homomorphic MAC. Sun et al. [17] combined MDB-tree and Merkle hash tree to realize. And later proposed public and private verification scheme based on bilinear-map accumulation [16].

3 Problem Formulation

3.1 System Model and Threat Model

System Model. The system model involves three entities: data owner, cloud server and data user, as shown in Fig. 1.

Data Owner: The data owner owns the plaintext dataset F locally. First data owner encrypts the plaintext document collection F by symmetric encryption algorithm, and generates the secure index tree T to improve the search efficiency. Later, outsource the index tree to the cloud server along with the encrypted document collection C. And the secret keys for document encryption and secure kNN algorithm are sent to the data user via a secure channel. When data owner wants to update the data on the cloud, update request will be sends to cloud.

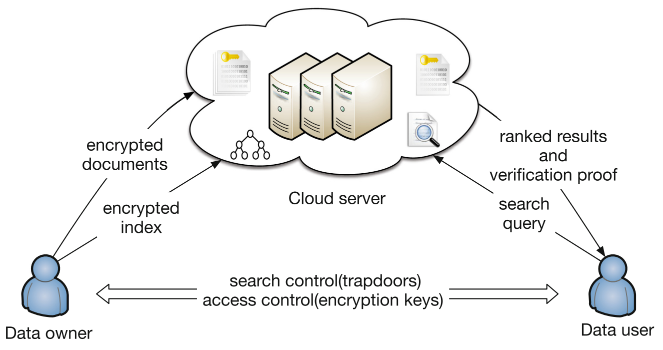

encrypted documents
encrypted index
Cloud server
ranked results and verification proof
search query
search control(trapdoors)
access control(encryption keys)
Data owner
Data user

Fig. 1. The architecture of our scheme.

Cloud Server: The cloud server is responsible for the data storage. The encrypted document collection and encrypted index tree from data owner are stored on the cloud. After receiving the search trapdoor from the data user, cloud server traverses the secure index tree to search for the relevant documents and sends back the top-k most similar results to the data user. Meanwhile the cloud server also sends back information to the data user for verification. When receiving the update request from data owner, the cloud server uses the information from data owner to update the document collection and index tree.

Data User: The data user who has access to the data can upload a search request consisting of some of keywords in the keyword dictionary. The data user uses the secret key to generate a query trapdoor and sends it to the cloud. When receiving the results from cloud server, data user can decrypts the results with the symmetric key locally. The verification proof returned can be used to verify the correctness, completeness and freshness of the results.

Threat Model. We assume the data user is authorized and trusted so that we don't consider the leakage in the data user side. Nor do we consider the leakage of secret key on the channel of key distribution. But the cloud server is assumed to be honest-but-curious. In other words, the cloud server will follow the processes honestly but will be curious to the content of data, keywords and other additional information. We mainly consider two threat models.

Known Ciphertext Model. In this model, the cloud server only knows encrypted information, specifically, the encrypted document collection C, the encrypted tree-based index I and the encrypted trapdoor.

Known Background Model. In this model, the cloud server knows additional backgrounds, such as the document frequency and keyword frequency. These information will be used to conduct statistical attack to infer the keywords in the query request.

3.2 Design Goals

To achieve a privacy-preserving, efficient, dynamic and verifiable multi-keyword ranked search scheme over encrypted data on the cloud, we propose the following design goals:

Search Efficiency. The time cost of search should be appropriate to the large amount of data. Tree-based structure is a great way to achieve it.

Dynamic Update. The scheme should support dynamic update, including insertion, deletion and modification.

Result Verifiable. The scheme can verify whether the returned results are what the data user want or not. (1) Correctness. The results should satisfy the query request and all originated from data owner with unmodified version. (2) Completeness. The results should contain all the search results which match the query request. (3) Freshness. The results should be the freshest and unmodified.

Privacy. The scheme can prevent the data or any other information from being analyzed by cloud server. (1) Data privacy. The server cannot recover the plaintext documents by analyzing the ciphertext. Cryptography is always used to protect the data. (2) Index and query privacy. The index and query are represented by vectors which contain the information of keywords such as the TF value in the index and the IDF value in the trapdoor which should be protected. (3) Keyword privacy. The cloud server could not make out the specific keywords. (4) Trapdoor unlinkability. The trapdoor need to be indistinguishable for the same query.

3.3 Notations

See Table 1.

Table 1. The notations in our scheme.

F	The plaintext document collection stored in data owner side, which contains N documents, and denoted as $F = \{f_1, f_2, \ldots, f_N\}$
W	The dictionary of n keywords shared between data owner and data user, denoted as $W = \{w_1, w_2, \ldots, w_n\}$
C	The encrypted document collection stored in cloud server side, denoted as $C = \{c_1, c_2, \ldots, c_N\}$
T	The unencrypted index tree generated from the document collection C
I	The encrypted index tree generated from tree T
Q	The query vector submitted by data user contains m keywords in W, denoted as $Q = \{q_1, q_2, \ldots, q_m\}$
TD	The trapdoor generated from the query vector Q and will upload to cloud server
R	The top-k encrypted document search results returned from cloud server for decryption and verification
PR	The plaintext document results decrypted by R

3.4 Preliminaries

Vector Space Model with TF $\times$ IDF. Vector space model is one of the most widely used models for information retrieval whose basic idea is to represent the document and query as a vector. Each dimension of the vector corresponds to a keyword, and the value is the weight of the keyword, which can be calculated using the $TF \times IDF$ mechanism [23]. The term frequency, TF, the frequency of the word in the document which reveals the significance of the word. The inverse document frequency, IDF, is the number of documents which contain the word among the document collection. And the $TF \times IDF$ algorithm uses the product of TF and IDF to measure the correlation between the keywords and document:

$$\begin{aligned} S &= \sum_{w_i \in Q} TF_{f_i,w_i} \times IDF_{w_i} \\ &= \sum_{w_i \in Q} \frac{ln(1+N_{f,w_i})}{\sqrt{\sum_{w_i \in W}(ln(1+N_{f,w_i}))^2}} \times \frac{ln(1+N/N_{w_i})}{\sqrt{\sum_{w_i \in W}(ln(1+N/N_{w_i}))^2}} \end{aligned} \tag{1}$$

where N_{f_i,w_i} is the number of keyword w_i in document f_i. N is the total number of documents in collection, N_{w_i} is the number of documents that contain the keyword w_i.

This method is intuitive, also the processing speed is fast. However, it neglects many other characteristics. Fully considering the effect of the word location and the document length on the weight, we optimize the algorithm by introducing them into $TF \times IDF$. The optimized $TF \times IDF$ is denoted as:

$$\begin{aligned} S &= \sum_{w_i \in Q} TF'_{f_i,w_i} \times IDF_{w_i} \\ &= \sum_{w_i \in Q} \frac{\frac{n_{w_i}}{L_{f_i}} \times \sum_1^k(\gamma_{f_j} \times tf_{j,w_i})}{\sqrt{\sum_{w_i \in W}(\frac{n_{w_i}}{L_{f_i}} \times \sum_1^k(\gamma_{f_j} \times tf_{j,w_i}))^2}} \times IDF_{w_i} \end{aligned} \tag{2}$$

where the document can be divided into k parts, γ_{f_j} is the weighting coefficient of the j^{th} part of the document, and the sum of the coefficient is 1, tf_{j,w_i} is the number of keyword w_i in the j^{th} part, n_{w_i} is the number of the keyword w_i, L_{f_i} is the length of the whole document f_i.

Tree-Based Index Construction. As shown in Fig. 2, we construct the indexes to a binary tree based on the Xia's scheme [22]. This tree-based index structure can greatly improve the efficiency of search.

In this structure, each node u in the tree is defined as:

$$u = (ID, P_l, P_r, D, h). \tag{3}$$

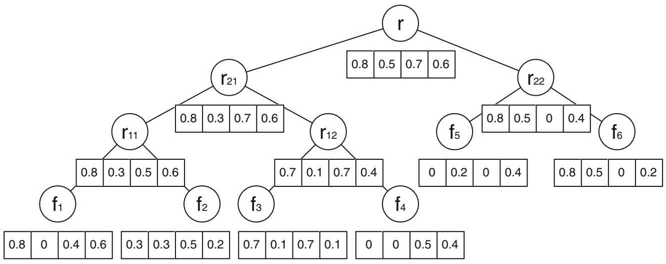


Fig. 2. Index tree.

where ID is the identity of the node, P_l and P_r is the pointers to the left and right child node, D is the index vector whose dimensions are the TF weight of keywords, and h is the hash value of node u for verification. Each leaf node is linked to a document. The building process is shown in Algorithm 1.

Algorithm 1. BuildTree(F)

```
Input: the plaintext documents $F$, the encrypted indexes $I$
Output: the index tree $T$
1: for each document $f_i$ in $F$ do
2:     Initial the leaf node. $u.ID = ID(f_i)$,$u.P_l = null$,$u.P_r = null$,$u.D[i] = TF_{f_i,w_i}$ for $i \in [1, n]$
3: end for
4: while the root node is not generated do
5:     Generate the parent node for each two nodes $u^{'}$ and $u^{''}$. $u.ID = ID(u)$,$u.P_l = u^{'}$,$u.P_r = u^{''}$,$u.D[i] = max\{u^{'}.D[i], u^{''}.D[i]\}$ for $i \in [1, n]$
6: end while
7: return the tree $T$
```

4 The Proposed Scheme

4.1 Detail Scheme

The detail scheme is as follows.

Algorithm 2. Search(u)

```
Input: index tree node $u$, trapdoor $TD$, threshold $TH$
Output: k documents $R$
1: compute the relevance score $S = u.D \cdot TD$
2: if the node u is not a leaf node then
3:     if $S > TH$ then
4:         $Search(u.P_l)$
5:         $Search(u.P_r)$
6:     end if
7: else
```

```
8:      if $S > TH$ then
9:          insert the node and the score into $r$
10:         if $length(R) > k$ then
11:             sort $R$ and delete the result with minimum score
12:             $TH = min\{R\}$
13:         end if
14:     end if
15: end if
16: return $R$
```

- $\{SK, sk\} \leftarrow Initial(1^l)$. This algorithm generates the secret keys for encrypting the documents, indexes and query. The data owner generates the secret key sk for encrypting and decrypting the documents, and the secret key SK for encrypting indexes. SK is composed of three elements, one $(n+U+1)$-bit vector as S, and two $(n+U+1) \times (n+U+1)$ invertible matrices as $\{M1, M2\}$, where U is a random number of dummy keywords to insert. Thus, $SK = \{S, M1, M2\}$.
- $C \leftarrow Enc(F, sk)$. The data owner uses a symmetric encryption algorithm such as AES to encrypt the plain document collection F.
- $I \leftarrow BuildIndex(F, SK)$. It is used to generate the encrypted index for each document. The data owner generates a n bit index vector for each document f_i in document collection F. Then, every vector is extended to $(n+U+1)$-bit vector p_i. The $(n+j)^{th}$ bit where $j \in [1, U]$, is set to a random number $\epsilon^{(j)}$. And the $(n+U+1)^{th}$ bit is set as 1. Then, call the algorithm $T \leftarrow BuildTree(F)$ to construct a index tree. Next, the index vector p_i on each node of the tree is split into two random vectors $\{p_i^{'}, p_i^{''}\}$ by the secret vector S for splitting. Namely, if $S[j] = 0$, we set $p_i^{'}[j] = p_i^{''}[j] = p_i[j]$; if $S[j] = 1$, we set $p_i^{'}[j]$ and $p_i^{''}[j]$ as random numbers and $p_i^{'}[j] + p_i^{''}[j] = p_i[j]$. The index is encrypted as $I_i = \{M_1^T p_i^{'}, M_2^T p_i^{''}\}$.
- $TD \leftarrow Trapdoor(Q, SK)$. The data user generates a n bit index vector for the search query Q, in which each dimensions are set to the IDF_{w_i} of the keywords w_i, and for other keywords, $Q[i] = 0$. Then, the query vector is extended to $(n+U+1)$-bit vector q. Choose a random number v out of U, the v random positions in $[n, n+U]$ are set to 1, others are set to 0. And the $(n+U+1)^{th}$ bit is set to a random number $t (t \in [0,1])$. Scale the first $n+U$-bit, denoted as $Q^{'}$, by a random number r, then the query $q = (rQ^{'}, t)$. Next, the query vector q is split into two random vectors $\{q^{'}, q^{''}\}$ by the secret vector S for splitting. Namely, if $S[j] = 1$, we set $q^{'}[j] = q^{''}[j] = q[j]$; if $S[j] = 0$, we set $q^{'}[j]$ and $q^{''}[j]$ as random numbers and $q^{'}[j] + q^{''}[j] = q[j]$. Then, the trapdoor is encrypted as $TD = \{M_1^{-1} q^{'}, M_2^{-1} q^{''}\}$.
- $R \leftarrow Search(I, TD)$. This algorithm uses indexes and trapdoor to calculate the similarity to get the top-k research results, showed in Algorithm 2. After receiving the trapdoor TD from the data user, the cloud server calculates the relevance score between TD and the index vector stored in each node to get

the top k relevant results. The relevance score is calculated as:

$$\begin{aligned} S &= I_i \cdot TD \\ &= (M_1^T p_i^{'}) \cdot (M_1^{-1} q^{'}) + (M_2^T p_i^{''}) \cdot (M_2^{-1} q^{''}) \\ &= p_i^{'} \cdot q^{'} + p_i^{''} \cdot q^{''} \\ &= p_i \cdot q \end{aligned} \tag{4}$$

– $PR \leftarrow Dec(R, sk)$. The data user uses the secret key sk transmitted from the data owner by a secret channel to decrypt the secret results R and get the plaintext results PR.

4.2 Result Verification

Algorithm 3. hash tree construction

```
1: for each leaf node do
2:     $u.h = hash(u.ID||\Phi(f_i))$   // $\Phi(f_i)$ means the content of the document.
3: end for
4: for each nonleaf node do
5:     $u.h = hash(u.ID||h_{p_l}||h_{p_r})$
6:     if the node is root node then // signature
7:         $\sigma_r = Sign(u.h||ts)$   // $ts$ is the timestamp
8:     end if
9: end for
```

Algorithm 4. minimum hash sub-tree

Input: returned results R, index tree T
Output: minimum hash sub-tree *mintree*

```
1: for each node $u$ in $R$ do
2:     insert $u$ into mintree
3:     while $u$ is not root node do
4:         insert $u$'s father node and $u$'s brother node into mintree
5:         $u = u.parent$
6:     end while
7: end for
8: return mintree
```

Algorithm 5. Verify

Input: minimum hash sub-tree *mintree*, returned results R

```
1: if the signature of root node is true then // freshness and authentic
2:     if verification of each node in mintree is true then   // authentic
3:         recompute the hash value of nodes in $R$
4:         if the value after recomputing = the value in mintree then
5:             re-search the mintree using the same trapdoor // correctness
6:             if the re-search result = $R$ then // correctness and completeness
```

```
7:                return True
8:            end if
9:        end if
10:    end if
11: end if
```

We refer to the Merkle tree to design our verification scheme. The data owner construct the hash tree based on the index tree using Algorithm 3. For example, in Fig. 2, the hash value of leaf node f_1 is $hash(f_1.ID||\Phi(f_1))$, and the non-leaf node $r_{11}.h = hash(f_1.h||f_2.h)$ and the root node $r.h = hash(r_{21}.h||r_{22}.h)$. And generate the signature of the root node by signature algorithm like RSA signature algorithm. Then cloud server will return necessary proof for verification with the results. The proof includes the signature of the root node σ_r and the minimum hash sub-tree *mintree* generated by Algorithm 4. In Fig. 2, if the returned result is f_2, the proof is $(\sigma_r, f_2, r_{11}, r_{21}, r, f_1, r_{12}, r_{22})$. After receiving the proof and results, data user verifies the results to be completeness, correctness and freshness by Algorithm 5.

4.3 Dynamic Update

Since the data stored at the cloud server may be deleted and modified, new documents may be added, the scheme should support dynamic update. There are two aspects should be take into consideration. One is the keywords dictionary. This can be settled by keeping some blank space in the document vector in advance. We have a premiss that the dictionary is relatively fixed and with small increments. Therefore, this process can satisfy most of the situations and the overhead is relatively low.

The other is the update of file collection which will influence both the encrypted index tree and the encrypted file collection. The data owner preserves a plaintext index tree locally, and generates sufficient information for updating in an encrypted way showed in Algorithm 6. Enlightened by the minimum hash sub-tree, we will neither need to re-construct whole encrypted index tree nor to proceed *BuildIndex* on whole tree, which will reduce the efficiency since this algorithm contains many matrix operation.

Algorithm 6. Update proof

```
Input: the update file $f_{upd}$
1: $flag = \{insert, delete, modify\}$
2: encrypt the file $f_{upd}$ to $c_{upd}$
3: if $flag = insert$ then
4:     insert the $c_{upd}$ into the leaf nodes
5: end if
6: if $flag = delete$ then
7:     search and set the node of $c_{upd}$ to $null$
8: end if
9: if $flag = modify$ then
```

```
10:     search and update the node of $c_{upd}$
11: **end if**
12: re-build a new index tree
13: construct minimum sub-tree $t_{upd}$ for $c_{upd}$ by Algorithm 4
14: **return** $\{t_{upd}, flag, c_{upd}\}$
```

After receiving the information for updating, the cloud server use the information u_{upd} to update the corresponding nodes in the index tree and document c_{upd} to update the file collection. This process is showed below.

Algorithm 7. Update

```
Input: updated file $c_{upd}$, sub-tree $t_{upd}$, update operation $flag$
 1: if $flag = insert$ then
 2:     insert the $c_{upd}$ into the Collection $C$
 3: end if
 4: if $flag = delete$ then
 5:     search and delete the document
 6: end if
 7: if $flag = modify$ then
 8:     search and replace the document to $c_{upd}$
 9: end if
10: replace corresponding nodes in $T$ to $t_{upd}$
```

5 Performance Analysis

In order to estimate the performance, we implement the scheme on real data set using C# language on a Windows 7 server with Inter(R) Core(TM) i5-6500 3.20 GHz.

For ease of experiment, we choose the formatted paper set as our study object for they have typically fixed format such as title, abstract, body, conclusion and preferences. The data set contains 4529 papers with 2964 keywords. We refer to the parameter setting under the plaintext in other works and set the parameter as shown in Table 2. We analyze our scheme from precision, security and efficiency.

Table 2. The parameter in data labelling.

γ_0	0.45	Title
γ_1	0.35	Abstract
γ_2	0.1	Body
γ_3	0.07	Conclusion
γ_4	0.03	References

5.1 Precision

Precision is the fraction of real retrieved documents among all the returned document. Since the similarity score of a document will be greatly influenced and randomized, the process will produce false positive results and reduce the precision of the results. The precision is defined as: $Precision = k^{'}/k$, where $k^{'}$ is the number of real documents and k is the number of returned top-k documents. The results are shown in Fig. 3.

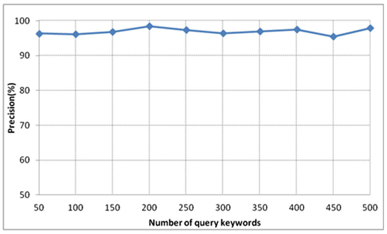


Fig. 3. The precision of search.

5.2 Security Analysis

Known Ciphertext Model. In this model, the cloud server can only obtain the encrypted document and encrypted index. The adversary distinguishes two documents mainly depending on the index generation $I \leftarrow BuildIndex(F, SK)$ and document encryption $C \leftarrow Enc(F, sk)$. The document vector is $(n + U + 1)$-bit. The first n-bit are the weight of the keywords. The U-bit are randomly chosen, and the last bit is set to 1.

For index generation, the documents are first split into two vectors and the number is set randomly if the number in S is "1". Assume the number of "1" in first n-bit and the last bit is μ_1 and each dimension of F is η_f, there are $(2^{\eta_f})^{\mu} \cdot (2^{\eta_f})^U$ possible values. Then the two vector are encrypted by two random $(n + U + 1) \times (n + U + 1)$-bit matrixes. Assume each elements in matrixes is η_M-bit, there are $(2^{\eta_M})^{(n+U+1)^2 \times 2}$ possible values for two matrixes. Thus the probability that indexes of two document are the same can be computed as follows:

$$\begin{aligned} P_d &= \frac{1}{(2^{\eta_f})^{\mu_1} \cdot (2^{\eta_f})^U \cdot (2^{\eta_M})^{(n+U+1)^2 \times 2}} \\ &= \frac{1}{2^{\mu_1 \eta_f + U\eta_f + 2\eta_M (n+U+1)^2}} \end{aligned} \qquad (5)$$

The larger μ_1, U, η_f and η_M are, the more difficult to distinguish. If we choose $\eta_f = 1024$, $P_d < 1/2^{1024}$ can be negligible. As a result, the encrypted indexes are indistinguishable.

For document encryption, since we choose the symmetric encryption with semantic secure, the encrypted documents are secure against known ciphertext model.

Known Background Model. In this model, the adversary can obtain some statistical information to infer the keywords or any other information.

The trapdoor is a $n + U + 1$-bit vector. The first n-bit represents whether the keyword exists in the query or not. U-bit dimension is included v out of U bit "1" and the other bits are "0". The last bit is set to a η_t-bit random number t. First the vector is scaled by a η_r-bit random number r which have 2^{η_r} possible values. Then the vector is split into two vector by a $(n + U + 1)$-bit S with μ_0 "0". Assume each dimension in first $(n + U)$-bit is η_q bits, there are $(2^{\eta_q})^{\mu_0} \cdot 2^{\eta_t}$. Then the two query vector are encrypted by two random matrixes. As a result the probability that two trapdoors are the same is computed as follows:

$$P_q = \frac{1}{2^{\eta_r} \cdot 2^{\eta_t} \cdot (2^{\eta_q})^{\mu_0}} \tag{6}$$

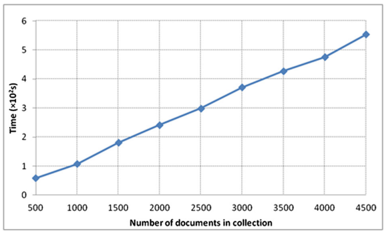
It can be proved to be indistinguishable by setting large number of η_r, η_t, η_q and μ_0. For example, if $\eta_r = 1024$, $P_q < 1/2^{1024}$ and can be negligible.

Privacy

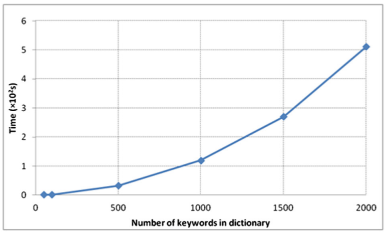
- *Data privacy.* Document collection is encrypted by a traditional symmetric encryption algorithm like AES which has been proved to be semantically secure.
- *Index and trapdoor privacy.* In our scheme, index I and trapdoor TD are encrypted by secure kNN algorithm. And the dummy keywords, the vector S for splitting and two matrixes M_1, M_2 for encrypting are all generated randomly which hide the plaintext in each dimension. As long as the secret key $SK = \{S, M1, M2\}$ is kept confidential, the cloud server can not identify the index or trapdoor by analyzing the ciphertext. It has been proved to be secure in the known ciphertext model [3].
- *Query unlinkability.* The random number r, t and v randomly chosen ϵ_i protect the search pattern and make the trapdoor indistinguishable even for the same search query. And thus, the similarity score will be different for each query and the cloud server cannot identify the relationship.
- *keyword privacy.* By introducing the random number ϵ_i in the index vector to randomize the similarity score, the keyword privacy can be well protected under known background model.

5.3 Efficiency Analysis

Index Construction. The index tree construction includes two process, building and encrypting by secure kNN algorithm. In building process, the tree is generated by all the documents in collection. The complexity of building is linear to the number of document $\mathcal{O}(N)$. Since encrypting process includes a split



(a)



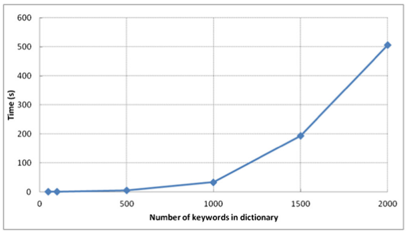
(b)

Fig. 4. The time cost of index tree construction. (a) For the different number of documents in collection with the fixed keyword dictionary $n = 1000$. (b) For the different number of keywords in dictionary with the fixed document collection $N = 1000$.

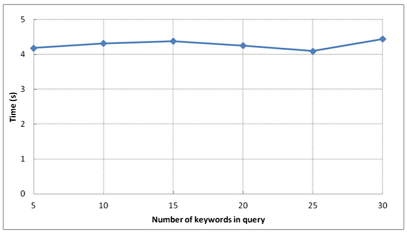
vector and two secret matrix, the complexity of encryption process depends on the number of keyword dictionary $\mathcal{O}(n^2)$. As a result, the time cost of index construction is mainly influenced by the number of document collection and keyword dictionary. The time cost are shown in Fig. 4. Since the process is one-time computation on the data owner, the time cost is acceptable.

Trapdoor Generation. The complexity of trapdoor generation depends on the split vector and secret matrix. Thus the complexity is related to the number of keyword dictionary $\mathcal{O}(n^2)$. And the number of keyword query has little influence of the time cost. The time cost is shown in Fig. 5.

Search. The search process can be briefly summarized as the product of each tree node and query vector. Thus the complexity of search mainly depends on the number of tree nodes and the number of keyword in dictionary. Actually, we don't need to compute the similarity score for every nodes. Based on the tree structure, we can compare the similarity score of nonleaf node with the threshold and eliminate the nodes which will apparently not be included in the final results. The time cost is shown in Fig. 6.

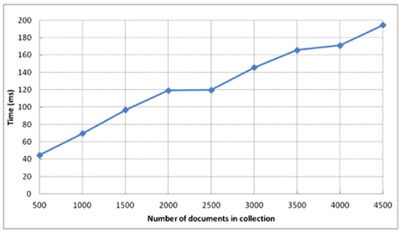


(a)

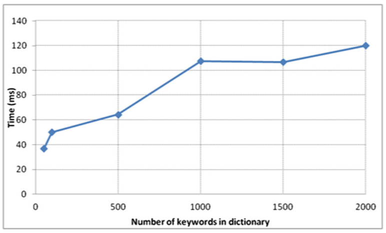


(b)

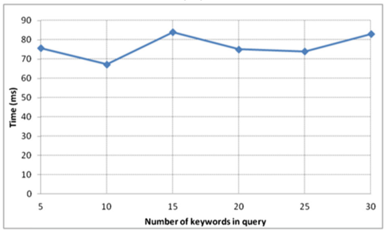
Fig. 5. The time cost of trapdoor generation. (a) For the different number of keywords in dictionary with the fixed keywords in query $m = 5$. (b) For the different number of keywords in query with the fixed keyword dictionary $n = 500$.



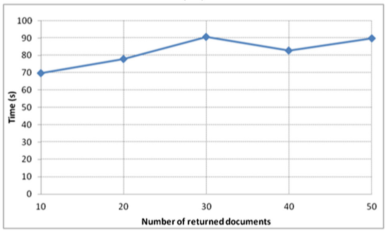
(a)



(b)



(c)



(d)

Fig. 6. The time cost of search. (a) For the different number of documents in collection with $n = 500$, $m = 5$ and $k = 20$. (b) For the different number of keywords in dictionary with $N = 1000$, $m = 5$ and $k = 20$. (c) For the different number of keywords in query with $N = 1000$, $n = 500$ and $k = 20$. (d) For the different number of return documents with $N = 1000$, $m = 5$ and $n = 500$.

6 Conclusion and Future Work

In this paper, we present a privacy-preserving, efficient ranked multi-keyword search scheme. We first focus on the formatted data set such as paper, project plan and so on. So that the keyword location and document length are introduced into the computation of TF value in our search scheme. A tree structure for index is designed to improve the efficiency of search. And we extend the functionality to implement verification and dynamic update. We give an analysis of the security against two threat model and apply our scheme on real paper set to analyze the performance (Table 3).

Table 3. The comparison among our scheme and related work.

Scheme	Verifiability	Dynamism	Construction	TF × IDF
Our paper	√	√	Binary tree	Location & length
[22]	×	√	KBB-tree	Tradition
[6]	×	×	MDB-tree & interest model	Tradition
[17]	√	×	MDB-tree	Tradition

This work still have further improvements. Inspired by Li et al. [13], the query can be extended to support operations with "AND", "OR", "NOT" by well designed parameters. And we can extend the scheme to support semantic-based sentence query. Furthermore, the nodes in the tree can be well designed such as clustering and partition to have a much better efficiency improvement.

Acknowledgments. This work is supported by the National Nature Science Foundation of China (NSFC) under grant 61379144 and 61572026, Open Foundation of State Key Laboratory of Cryptology (No: MMKFKT201617) and the Foundation of Science and Technology on Information Assurance Laboratory.

References

1. Boneh, D., Di Crescenzo, G., Ostrovsky, R., Persiano, G.: Public key encryption with keyword search. In: Cachin, C., Camenisch, J.L. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 506–522. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-24676-3_30
2. Boneh, D., Waters, B.: Conjunctive, subset, and range queries on encrypted data. In: Vadhan, S.P. (ed.) TCC 2007. LNCS, vol. 4392, pp. 535–554. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-70936-7_29
3. Cao, N., Wang, C., Li, M., Ren, K., Lou, W.: Privacy-preserving multi-keyword ranked search over encrypted cloud data. IEEE Trans. Parallel Distrib. Syst. **25**(1), 222–233 (2013). https://doi.ieeecomputersociety.org/10.1109/TPDS.2013.45
4. Chang, Y.-C., Mitzenmacher, M.: Privacy preserving keyword searches on remote encrypted data. In: Ioannidis, J., Keromytis, A., Yung, M. (eds.) ACNS 2005. LNCS, vol. 3531, pp. 442–455. Springer, Heidelberg (2005). https://doi.org/10.1007/11496137_30
5. Chen, C., et al.: An efficient privacy-preserving ranked keyword search method. IEEE Trans. Parallel Distrib. Syst. **27**(4), 951–963 (2016)
6. Fu, Z., Ren, K., Shu, J., Sun, X., Huang, F.: Enabling personalized search over encrypted outsourced data with efficiency improvement. IEEE Trans. Parallel Distrib. Syst. **27**(9), 2546–2559 (2016)
7. Fu, Z., Wu, X., Guan, C., Sun, X., Ren, K.: Towards efficient multi-keyword fuzzy search over encrypted outsourced data with accuracy improvement. IEEE Trans. Inf. Forensics Secur. **11**(12), 2706–2716 (2017)
8. Goh, E.J.: Secure indexes. Cryptology ePrint Archive, Report 2003/216 (2003). https://eprint.iacr.org/2003/216
9. Golle, P., Staddon, J., Waters, B.: Secure conjunctive keyword search over encrypted data. In: Jakobsson, M., Yung, M., Zhou, J. (eds.) ACNS 2004. LNCS, vol. 3089, pp. 31–45. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-24852-1_3
10. Kamara, S., Papamanthou, C.: Parallel and dynamic searchable symmetric encryption. In: Sadeghi, A.-R. (ed.) FC 2013. LNCS, vol. 7859, pp. 258–274. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39884-1_22
11. Kamara, S., Papamanthou, C., Roeder, T.: Dynamic searchable symmetric encryption. In: ACM Conference on Computer and Communications Security, pp. 965–976 (2012)

12. Lai, R.W.F., Chow, S.S.M.: Parallel and dynamic structured encryption. In: Deng, R., Weng, J., Ren, K., Yegneswaran, V. (eds.) SecureComm 2016. LNICST, vol. 198, pp. 219–238. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-59608-2_12
13. Li, H., Yang, Y., Luan, T., Liang, X., Zhou, L., Shen, X.: Enabling fine-grained multi-keyword search supporting classified sub-dictionaries over encrypted cloud data. IEEE Trans. Dependable Secur. Comput. **13**(3), 312–325 (2016)
14. Li, J., Wang, Q., Wang, C., Cao, N., Ren, K., Lou, W.: Fuzzy keyword search over encrypted data in cloud computing. In: Conference on Information Communications, pp. 441–445 (2010)
15. Song, D., Wagner, D., Perrig, A.: Practical techniques for searches on encrypted data. In: Proceeding 2000 IEEE Symposium on Security and Privacy, SP 2000, pp. 44–55 (2000)
16. Sun, W., Liu, X., Lou, W., Hou, Y.T., Li, H.: Catch you if you lie to me: efficient verifiable conjunctive keyword search over large dynamic encrypted cloud data. In: 2015 IEEE Conference on Computer Communications (INFOCOM), pp. 2110–2118, April 2015
17. Sun, W., et al.: Verifiable privacy-preserving multi-keyword text search in the cloud supporting similarity-based ranking. IEEE Trans. Parallel Distrib. Syst. **25**(11), 3025–3035 (2014)
18. Wan, Z., Deng, R.H.: VPsearch: achieving verifiability for privacy-preserving multi-keyword search over encrypted cloud data. IEEE Trans. Dependable Secur. Comput. **PP**(99), 1 (2017)
19. Wang, B., Song, W., Lou, W., Hou, Y.T.: Inverted index based multi-keyword public-key searchable encryption with strong privacy guarantee. In: Computer Communications, pp. 2092–2100 (2015)
20. Wang, C., Cao, N., Li, J., Ren, K., Lou, W.: Secure ranked keyword search over encrypted cloud data. In: Proceedings of the 2010 IEEE 30th International Conference on Distributed Computing Systems, pp. 253–262 (2010). https://doi.org/10.1109/ICDCS.2010.34
21. Wang, Q., He, M., Du, M., Chow, S.S., Lai, R.W., Zou, Q.: Searchable encryption over feature-rich data. IEEE Trans. Dependable Secur. Comput., 1 (2016), http://doi.ieeecomputersociety.org/10.1109/TDSC.2016.2593444
22. Xia, Z., Wang, X., Sun, X., Wang, Q.: A secure and dynamic multi-keyword ranked search scheme over encrypted cloud data. IEEE Trans. Parallel Distrib. Syst. **27**(2), 340–352 (2016). http://doi.ieeecomputersociety.org/10.1109/TPDS.2015.2401003
23. Zobel, J., Moffat, A.: Exploring the similarity space. ACM SIGIR Forum **32**(1), 18–34 (1998)