Hindawi
Security and Communication Networks
Volume 2020, Article ID 7307315, 15 pages
https://doi.org/10.1155/2020/7307315

Research Article

A Multibranch Search Tree-Based Multi-Keyword Ranked Search Scheme over Encrypted Cloud Data

Hua Dai,[1,2] **Xuelong Dai**,[1] **Xiao Li**,[1] **Xun Yi**,[3] **Fu Xiao**,[1] **and Geng Yang**[1,2]

[1]*Nanjing University of Posts and Telecommunications, Nanjing 210023, China*
[2]*Jiangsu Security and Intelligent Processing Lab of Big Data, Nanjing 210023, China*
[3]*Royal Melbourne Institute of Technology University, Melbourne 3001, Australia*

Correspondence should be addressed to Hua Dai; daihua@njupt.edu.cn

Received 17 June 2019; Revised 4 December 2019; Accepted 19 December 2019; Published 23 January 2020

Academic Editor: Leandros Maglaras

In the interest of privacy concerns, cloud service users choose to encrypt their personal data before outsourcing them to cloud. However, it is difficult to achieve efficient search over encrypted cloud data. Therefore, how to design an efficient and accurate search scheme over large-scale encrypted cloud data is a challenge. In this paper, we integrate bisecting k-means algorithm and multibranch tree structure and propose the α-filtering tree search scheme based on bisecting k-means clusters. The novel index tree is built from bottom-up, and a greedy depth first algorithm is used for filtering the nonrelevant document cluster by calculating the relevance score between the filtering vector and the query vector. The α-filtering tree can improve the efficiency without the loss of search accuracy. The experiment on a real-world dataset demonstrates the effectiveness of our scheme.

1. Introduction

Cloud computing is a new model in IT enterprise which can offer high-quality calculation, storage, and application capacity. The cloud customers choose to outsource their local data and computation to the cloud server for minimizing the data maintenance cost. Thus, to protect users' privacy and achieve efficient and precise data retrieving from the cloud server has become the focus of recent works.

The traditional way to protect data privacy is to encrypt the original data. However, this is a very challenging task for data utilization. The search schemes based on ciphertext [1–7] can guarantee the data privacy but the search algorithms have high time and space complexity, which are not suitable for cloud data retrieval. To solve this problem, researchers proposed a series of searchable encryption schemes [8–13] based on the theory of cryptography. These encryption schemes either do not have high-accuracy retrieval results [8–10, 12] or cost a lot of time and space overhead [8, 11]. Therefore, it is a necessity to design an efficient and useable search scheme.

In this paper, we propose an α-filtering tree search scheme based on bisecting k-means clusters, which achieves efficient multi-keyword ranked search over encrypted cloud data. We use vector space model and TF-IDF model to build the keyword dictionary and transform the documents and keywords into "points" in a multidimensional space that can be described by vectors, and then we used the secure inner product to encrypt the document vectors and query vectors. The relevance scores between the document vectors and query vectors are used to obtain the top-k most relevant documents.

Our paper's main contributions are summarized as follows:

(i) We integrate the bisecting k-means algorithm and a multibranch tree structure where the bisecting k-means algorithm is used to improve the cluster accuracy, and we propose an α-filtering tree search scheme based on the bisecting k-means clusters.

(ii) We propose a greedy depth first algorithm to achieve searches on the α-filtering tree, which improves the multi-keyword search efficiency. By

adopting the secure inner product encryption scheme, we achieve the privacy-preserving ranked search on the encrypted α-filtering index tree.

(iii) We perform the experiment on a real-world dataset and compare with existing schemes in terms of retrieval efficiency and index storage usage. The result shows that our scheme is superior in search efficiency and storage usage.

The rest of the paper is organized as follows: Section 2 introduces the related work, Section 3 introduces the main background knowledge, and Section 4 gives a brief introduction to our system model, threat model, and design goals. The constructions of the α-filtering tree and search algorithm are presented in Section 5. Sections 6 and 7 give the experiment result and its analyses. Finally, the conclusion is given in Section 8.

2. Related Work

Searchable encryption schemes implement keyword searches over encrypted outsourced data, which allow users to store their personal data on the cloud server without privacy concerns. Recently, an increasing number of scholars conduct research in this area. We discuss the related work on the development of searchable encryption schemes' performance and function.

2.1. Single-Keyword Searchable Encryption. Song et al. [14] first proposed a symmetric encryption search scheme, and they encrypted each keyword in the document set separately and searched the entire data set by sequential scanning. Thus, the search time of the scheme is linear with the overall size of the document set. Goh [15] proposed a searchable encryption scheme based on Bloom filter. They achieved search efficiency, that is to say, the calculation overhead is not related to the number of documents in the dataset. However, due to the probability of a false positive for the Bloom filter, the cloud server may return documents that do not contain search keywords. The scheme in Chang and Mitzenmacher [16] used two indexes. The first index is to store and manage a premade dictionary by the user. The second requires twice the interactions between the user and the cloud server, which affects the user experience but it can achieve the same search efficiency as Goh [15]. Curtmola et al. [17] adopted two novel search schemes SSE-1 and SSE-2. SSE-1 is used to prevent chosen-keyword attacks (CKA1), and SSE-2 is against adaptive chosen-keyword attacks (CKA2). Their schemes' search time cost is proportional to the number of keywords retrieved. Boneh et al. [18] adopted a searchable encryption structure that allows everyone to store their data with the public key. But their scheme needs large amount of calculation.

2.2. Multi-Keyword Search Schemes. Multi-keyword searchable encryption allows the user to submit multiple search keywords to retrieve the most relevant documents. They can be further classified into ranked search and traditional search. In traditional search, most schemes are conjunctive keyword search which returns all the documents containing the search keywords, and conjunctive-subset keyword search which returns the documents containing the keyword subset. However, traditional search is not suitable for the ranked search. Cao et al. [8] first achieved a privacy-preserving multi-keyword ranked search scheme. In their scheme, documents and search keywords are described by the dictionary-scale vectors. The scheme uses coordinate matching to rank the documents. Since the weights of different keywords in documents are not considered, the retrieval result obtained by the scheme lacks accuracy, and the search time of the scheme is linear with the scale of the dataset. Sun et al. [9] proposed a novel multi-keyword ranked scheme; they used TF-IDF vector space model and cosine distance measurement to build an index tree structure. The experiment shows that their scheme is more efficient than linear search but lacked accuracy. Orencik et al. [10] adopted Locality-Sensitive Hashing to cluster the similar documents, but their ranked search result is also not accurate. Xia et al. [11] adopted the vector space model and KBB tree to build a dynamic multi-keyword ranked search scheme, which more precisely obtains the ranked result. However, as the scale of the documents increases, the index tree space cost is large, and the pruning effect of the search algorithm is also reduced, resulting in a decrease in search efficiency.

To enhance searchable encryptions' usability and functionality, many schemes that support fuzzy keyword search [19–23], conjunctive keyword search [3, 24–26], and similarity search [27–30] have also been presented. The dynamic scheme can support updates on the dataset, which largely enhances searchable encryptions' usability. The first dynamic searchable encryption scheme is proposed by van Liesdonk et al. [31], which supports a limited number of updates. After their work, many dynamic searchable encryption schemes are proposed [32–35]. The verifiable scheme can check the integrity of search results when the cloud server is not honest. Many researches are conducted to support verifiable searches in [26, 36, 37]. To extend the searchable encryption scheme to support other data types like multimedia data, some research works were also proposed [38, 39].

In Xia et al.'s work [11], they presented an efficient search index tree to obtain the search result. However, as the scale of the document increases, the index tree space cost is large and the pruning effect of the search algorithm is also reduced, resulting in a decrease in search efficiency. Thus, we proposed a multibranch index tree to overcome this problem. By adopting the clustering algorithm over the document set, we can further increase the search efficiency. Moreover, the multibranch tree can also save the space cost for the index tree.

3. Notations and Background Knowledge

3.1. Notations. For simplicity, we defined our mainly used symbols as follows:

(i) W—the keyword dictionary contains n keywords, denoted as $W = \{w_1, w_2, \ldots, w_n\}$

(ii) D—the plaintext documents set contains m documents, denoted as $D = \{d_1, d_2, \ldots, d_m\}$

(iii) F_V—the plaintext document vector

(iv) $\widetilde{F}_V$—the encrypted form of F_V

(v) I—the α-filtering index tree

(vi) $\widetilde{I}$—the encrypted form of I

(vii) Q—the set of query keywords, $Q = \{w_1, w_2, \ldots, w_q\}$

(viii) V_Q—the Q's query vector

(ix) T_Q—the query trapdoor

(x) RL—the ranked search result list

(xi) μ—the threshold of the maximum number of documents in an atom cluster

(xii) α—the threshold of the maximum number of child nodes in a node of the α-filtering tree

3.2. Vector Space Model. Among many information retrieval models, the vector space model is the most popular method of relevance measurement and we adopt the TF-IDF model for feature extraction. It is widely used in plaintext multikeyword retrieval. TF (term frequency) refers to the word frequency, that is, the number of occurrences of the keyword w in the document f divided by the total number of words $|f|$ contained in the document f. IDF (inverse document frequency) indicates the inverse document frequency, that is, the number of documents divided by the number of documents containing the keyword.

The keyword dictionary is first generated by filtering the stop words form all the words contained in the document set D. Then, the document vector F_V and the query vector q are generated according to the keyword dictionary W. The dimension of F_V and q is equal to the scale of the keyword dictionary; each dimension represents a corresponding keyword w_i. The value of each dimension in F_V means the normalized TF value and normalized IDF value in q. The TF value and the IDF value of the keyword w_i are calculated as follows:

$$\begin{aligned} \mathrm{TF}_{f,w} &= \frac{\mathrm{TF}'_{f,w}}{\sqrt{\sum_{w \in W} \left(TF_{f,w}\right)^2}}, \\ \mathrm{IDF}_w &= \frac{\mathrm{IDF}'_w}{\sqrt{\sum_{w \in W} \left(\mathrm{IDF}'_w\right)^2}}, \end{aligned} \tag{1}$$

where $\mathrm{IDF}'_w = \ln(1 + N/N_w)$ and $\mathrm{TF}'_{f,w} = N_{f,w}/|f|$.

3.3. Relevance Measurement. The inner product operation is performed by two equal-length vectors, and the relevance between two vectors is quantized by the inner product score. The larger the score, the higher the relevance between the two vectors. The relevance score is calculated as follows:

$$\mathrm{Score}\left(F_V, V_Q\right) = F_V \cdot V_Q. \tag{2}$$

We make the following instructions about equation (2):

(i) If F_V is a document vector and V_Q is a search vector, Score (F_V, V_Q) is the relevance score between the document and the search keywords.

(ii) IF F_V is a filtering vector of the index tree node and V_Q is a search vector, Score (F_V, V_Q) is the relevance score between the upper bound vector of the documents stored in this node and the search keywords.

3.4. Bisecting k-Means Cluster. In data mining, the bisecting k-means algorithm is a cluster analysis algorithm. By selecting 2 initial centroids in a bisecting k-means algorithm, each point is assigned to the nearest centroid in turn, and the points that are assigned to the same centroid form a cluster. The centroids of each cluster are continually updated by different points assigned to the cluster, assignments and updates are repeated until the clusters no longer change, and then the clustering algorithm is completed. We use the cosine distance to measure the distance from the point to the centroid, which is defined in the following equation:

$$\mathrm{sim}(x, y) = \cos\theta = \frac{\overrightarrow{x} \cdot \overrightarrow{y}}{\|\overrightarrow{x}\| \cdot \|\overrightarrow{y}\|}, \tag{3}$$

where $\overrightarrow{x}$ is the point's vector, $\overrightarrow{y}$ is the centroid's vector, and $\|\overrightarrow{x}\|$ and $\|\overrightarrow{y}\|$ are the norms of $\overrightarrow{x}$ and $\overrightarrow{y}$.

3.5. Secure Inner Product Operation. The special matrix encryption proposed in [8] can achieve privacy-preserving vector inner product. Assuming that p and q are two n-dimensional vectors, the user encrypted them to $\widetilde{p}$ and $\widetilde{q}$ by calculating $M^T p$ and $M^{-1} q$, where M is a random $n \times n$ invertible matrix. Therefore, we can get the inner product of the original vectors only by the inner product of their encrypted form $\widetilde{p} \cdot \widetilde{q}$ as follows:

$$\begin{aligned} \widetilde{p} \cdot \widetilde{q} &= \left(M^T p\right)^T \cdot \left(M^{-1} q\right) \\ &= p^T M M^{-1} q \\ &= p \cdot q. \end{aligned} \tag{4}$$

4. Model and Problem Formulation

4.1. System Model. In this paper, there are 3 entities in our system model: data owner, data user, and cloud server as shown in Figure 1. These three entities collaborate as follows.

The data owner has the local dataset D and wants to outsource them in secure form to the cloud server while still providing the search service for users. In our scheme, it first generates the searchable index tree I according to D. Then, it uses the secure key to encrypt both D and I to its encrypted

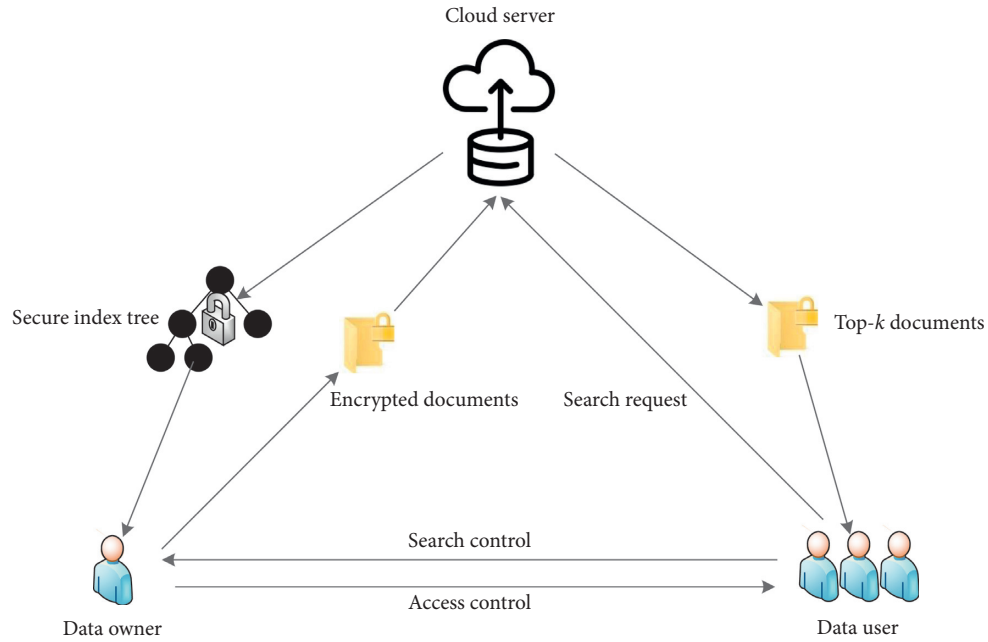


Figure 1: System model.

form $\tilde{D}$ and $\tilde{I}$. After that, it shares the secure key with the data user through the access control and outsources $\tilde{D}$ and $\tilde{I}$ to the cloud server.

The cloud server provides both storage service and search service. It stores the secured index tree $\tilde{I}$ and encrypted document set $\tilde{D}$. After it receives the search trapdoor T_Q from the data user, it performs the secure search by using $\tilde{I}$ and returns the search result to the data user.

The data user is the authorized one to access the document set. It generates the search trapdoor T_Q with search keywords Q through the proposed search scheme and sends T_Q as the search request to the cloud server. After receiving the search result, it uses the secure key to decrypt the encrypted documents and get the plaintext documents.

4.2. Threat Model. We adopt the same "honest-but-curious" threat model as the current work [8, 9, 11, 40, 42]. That is to say, the cloud server follows the user's instruction honestly and precisely, but it could curiously analyze the received data to obtain additional information about the dataset. Two threat models were proposed by Cao et al. [8] and are adopted in our work as follows:

Known Cyphertext Model: the cloud server could access the cyphertext dataset, the encrypted index tree, and the search trapdoor, and thus the cloud server can conduct the cyphertext-only attack.

Known Background Model: the cloud server could have more dataset-related information than the known cyphertext model in this stronger model. The cloud server can have statistical information about the relation between the search trapdoor and the search result. Then, it could infer or recognize some of the search keywords in the trapdoor by the additional information it has.

4.3. Design Goals. To ensure the privacy, efficiency, and accuracy in the multi-keyword ranked search over encrypted cloud data, our system design should meet these requirements as follows:

Search Efficiency: compared with other multi-keyword search schemes, the proposed search scheme should be superior in efficiency than others.

Search Accuracy: the proposed search scheme should guarantee the accuracy of the search result.

Privacy Persevering: the proposed scheme should ensure the privacy of the document privacy, index privacy, trapdoor privacy, trapdoor unlinkability, and keyword privacy in the search process.

5. Index and Search Algorithm

In this section, we mainly discuss the index construction method and search method based on the index tree and then we give the corresponding algorithms. We first construct a document atom cluster list by using the bisecting k-means algorithm. Then, based on the generated atom cluster list, we build the α-filtering tree and then propose a corresponding greedy depth first search algorithm for multi-keywords ranked search.

5.1. Atom Cluster List Generation Algorithm. Considering the document set D as the input raw cluster, we use the bisecting k-means algorithms to perform top-down bisecting clustering until all the generated subclusters contain less than μ documents in Algorithms 1 and 2, and thus a binary clustering tree is built as shown in Algorithm 2. Here, μ is the given threshold for clustering. Then, we traverse the leaf clusters in the generated binary clustering tree, and the atom

cluster list L is constructed in Algorithm 3, which is used for building the α-filtering index tree.

Definition 1. Atom Cluster. The leaf clusters in the binary tree generated by Algorithm 1 are the atom clusters, where the number of documents in each atom cluster is no more than μ.

Theorem 1. *Assuming that the list of the atom clusters generated by Algorithm 1 is $L = \{C_1, C_2, \ldots, C_t\}$, we have the following properties*

(1) $1 \le |C_i| \le \mu$

(2) $C_1 \cup C_2 \cup \ldots \cup C_t = D$

We illustrate the generation process of the atomic cluster list L in Algorithms 1–3 by an example. We assume that the document set is $D = \{d_1, d_2, \ldots, d_{15}\}$ and $\mu = 3$. The first round of bisecting clustering is performed on D, and two subclusters are generated as shown in Figure 2. With the same process, the second layer's and the third layer's subclusters are all sequentially divided into two clusters, and the subcluster stops clustering when the number of documents contained in the subcluster is less than or equal to 3. Finally, a binary clustering tree is formed, where the leaf nodes are $C_1 = \{d_1, d_2, d_3\}$, $C_2 = \{d_4, d_5, d_6\}$, $C_3 = \{d_9, d_{10}\}$, $C_4 = \{d_{11}, d_{12}\}$, $C_5 = \{d_7, d_8\}$, and $C_6 = \{d_{13}, d_{14}, d_{15}\}$, as shown in Figure 2. Then, the algorithm traverses the leaf nodes of the binary clustering tree in the middle order and then the atom cluster list $L = \{C_1, C_2, C_3, C_4, C_5, C_6\}$ is generated.

5.2. α-Filtering Tree

Definition 2. Upper Bound Vector. For a n-dimensional vector set $V = \{v_1, v_2, \ldots, v_m\}$, V's upper bound vector is a n-dimensional vector $V_{\mathrm{UB}} = UpBound\{v_1, v_2, \ldots, v_m\}$, where $V_{\mathrm{UB}}[i] = \max\{v_1[i], v_2[i], \ldots, v_m[i]\}$, $i = 1, 2, \ldots, n$.

Definition 3. α-Filtering Tree. A node u in the α-filtering tree is a triple, which is denoted as

$$u = < \mathrm{FV}, \mathrm{PL}, \mathrm{DC} >, \tag{5}$$

where u·FV is a n-dimensional filtering vector, u·PL is a child node pointer which have at most α pointers, and u·DC stores documents when u is a leaf node.

(1) If u is a leaf node, then u·PL = ∅, u·DC = $\{d_1, d_2, \ldots, d_x\}$ which corresponds to an atom cluster where $x \le \mu$, and u·FV = $UpBound\{\mathrm{FV}_1, \mathrm{FV}_2, \ldots, \mathrm{FV}_x\}$, where FV_i is the document vector of d_i

(2) If u is a non-leaf node, then u·DC = ∅, u·PL has at most α pointers, i.e., u·PL = $\{u$·PL [1], u·PL [2], …, u·PL$[g]\}$, where u·PL$[i]$ points to the i^{th} child node and $1 \le i \le g \le \alpha$, u·FV = $UpBound\{u$·PL [1]·FV, u·PL [2]·FV, …, u·PL$[g]$·FV$\}$ which is the upper bound vector of the filtering vectors of the child nodes in u·PL

We give the construction procedures of the α-filtering tree in Algorithm 4.

Algorithm 4 builds the α-filtering tree with the atom cluster list. Tree nodes are created during each round processing of steps 8-21. The original atom cluster list is treated as the first child node list (CNL). In each round, α nodes are fetched from CNL once a time and a parent node is created to have these nodes and added into the parent node list (PNL). After all the nodes in CNL have been fetched and then we have the complete parent node list (PNL) in this round. If we have more than 1 node in PNL, then we move all nodes in PNL to CNL. Otherwise, the only node in PNL is the root of the generated index tree.

Theorem 2. *The height of an α-filtering tree with t leaf node is* $\lceil \log_\alpha t \rceil + 1$.

Proof. We assume that the length of the atom cluster list L is t, that is, the number of leaf nodes of the α-filtering tree is t. According to Algorithm 4, after the $1^{\mathrm{st}}, 2^{\mathrm{nd}}, \ldots, x^{\mathrm{th}}$ rounds of processing, the number of current generated parent nodes becomes $\lceil t/\alpha \rceil, \lceil t/\alpha^2 \rceil, \ldots, \lceil t/\alpha^x \rceil$. When the number of current generated parent nodes is 1, the construction of the α-filtering tree is finished, so there is $\lceil t/\alpha^x \rceil = 1$. Then, we deduce $x = \lceil \log_\alpha t \rceil$. Since the height of each tree is increased by 1 for each merge and the initial height of the tree is 1, the height of the α-filtering tree with t leaf node is $\lceil \log_\alpha t \rceil + 1$. □

5.3. Multi-Keyword Ranked Search Algorithm.

By adopting the greedy depth-first search algorithm on the tree index, we can obtain the top-k documents efficiently by preexcluding the subtree that certainly does not contain any search result documents. We give the greedy depth-first search algorithm in this section.

Definition 4. For a query Q whose vector is V_Q and two nodes u and u', if Score (V_Q, u·FV) ≥ Score (V_Q, u'·FV), then u has higher or equal relevance score with Q than u' which is denoted as

$$u \rhd u'. \tag{6}$$

Theorem 3. *We assume that $u = <FV, PL, DC>$ is a non-leaf node in the α-filtering tree and u·PL stores g child nodes, i.e., u·PL = $\{u$·PL [1], u·PL [2], …, u·PL$[g]\}$ and $1 \le g \le \alpha$. For a query Q, we have*

$$\forall u' \in u \cdot \mathrm{PL} \longrightarrow u \rhd u'. \tag{7}$$

Proof. To prove $\forall u' \in u \cdot \mathrm{PL} \longrightarrow u \rhd u'$, that is to prove

$$\mathrm{Score}(V_Q, u \cdot \mathrm{FV}) \ge \max\{\mathrm{Score}(V_Q, u \cdot \mathrm{PL}[1] \cdot \mathrm{FV}), \mathrm{Score}(V_Q, u \cdot \mathrm{PL}[2] \cdot \mathrm{FV}), \ldots, \mathrm{Score}(V_Q, u \cdot \mathrm{PL}[g] \cdot \mathrm{FV})\}.$$

Because every elements in an n-dimensional filtering vector u·FV are generated by the following equation:

Input:
The document set, D;
The threshold of the maximum number of documents in an atom cluster, μ;
Output:
An atom cluster list, L;
(1) Create a root cluster node r which has all documents of D;
(2) *GenBiSectingTree* (r, μ);
(3) *InOrder* (r, L);
(4) return L;

Algorithm 1: *GenAtomClusterList* (D, μ).

Input: A cluster node having a set of documents, c; μ;
Output: The root of the generated binary clustering tree;
(1) **if** $|c| > \mu$ **then**
(2) Perform the bisecting k-means algorithm on c, and two bisected clusters, c_l and c_r, are generated, where $c = c_l \cup c_r$. Then, set $c \longrightarrow lchild = c_l$ and $c \longrightarrow rchild = c_r$, where *lchild* and *rchild* are the left and right child cluster nodes of c.
(3) *GenBiSectingTree* ($c \longrightarrow lchild$, μ);
(4) *GenBiSectingTree* ($c \longrightarrow rchild$, μ);
(5) **end if**

Algorithm 2: *GenBiSectingTree* (c, μ).

Input: c, μ;
Output: L;
(1) **if** c is not empty **then**
(2) *InOrder* ($c \longrightarrow lchild$, μ, L);
(3) **if** $|c| < \mu$ **then**
(4) Append c to L;
(5) **end if**
(6) *InOrder* ($c \longrightarrow rchild$, μ, L);
(7) **end if**

Algorithm 3: *InOrder* (c, μ, L).

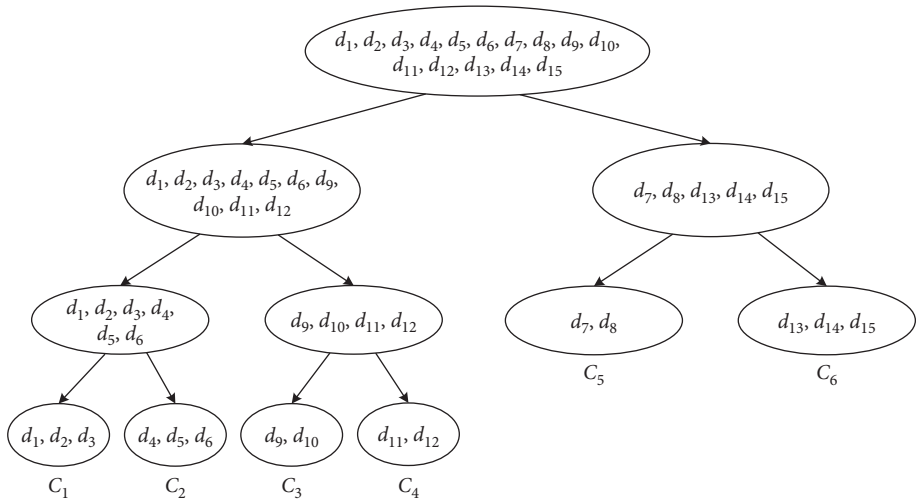


Figure 2: An example of the generation process of L.

Input: L; the threshold of the maximum number of child nodes, α
Output: the α-filtering tree, I
(1) **for** each C_i in L **do**
(2) Create a leaf node c for C_i. Assuming that $C_i = \{d_1, d_2, \ldots, d_x\}$, then $u{\cdot}\mathrm{PL} = \varnothing$, $u{\cdot}\mathrm{DC} = C_i$, $u{\cdot}\mathrm{FV} = UpBound\{\mathrm{FV}_1, \mathrm{FV}_2, \ldots, \mathrm{FV}_x\}$
(3) Add u into CNL; //CNL is a variable of the child node list and PNL is a variable of the newly generated parent node list
(4) **end for**
(5) **if** $|\mathrm{CNL}| < 1$ **then**
(6) **return** the only node left in CNL which is the root of the generated α-filtering tree
(7) **end if**
(8) **while** CNL is not empty **do**
(9) **while** CNL is not empty **do**
(10) **if** $|\mathrm{CNL}| \geq \alpha$ **then**
(11) Fetch the first α nodes $\{u_1, u_2, \ldots, u_\alpha\}$ from CNL and create a parent node u, where $u{\cdot}\mathrm{PL}[1] = u_1$, $u{\cdot}\mathrm{PL}[2] = u_2$, $\ldots$, $u{\cdot}\mathrm{PL}[\alpha] = u_\alpha$, $u{\cdot}\mathrm{FV} = UpBound\{u_1{\cdot}\mathrm{FV}, u_2{\cdot}\mathrm{FV}, \ldots, u_\alpha{\cdot}\mathrm{FV}\}$ and $u{\cdot}\mathrm{DC} = \varnothing$; then add u to PNL
(12) **else**
(13) Assuming that $\mathrm{CNL} = \{u_1, u_2, \ldots, u_g\}$, $1 \leq g < \alpha$, fetch all the nodes from CNL and create a parent node u where $\{u{\cdot}\mathrm{PL}[1] = u_1, u{\cdot}\mathrm{PL}[2] = u_2, \ldots, u{\cdot}\mathrm{PL}[g] = u_g\}$, $u{\cdot}\mathrm{FV} = UpBound\{u_1{\cdot}\mathrm{FV}, u_2{\cdot}\mathrm{FV}, \ldots, u_g{\cdot}\mathrm{FV}\}$ and $u{\cdot}\mathrm{DC} = \varnothing$; then add u to PNL
(14) **end if**
(15) **end while**
(16) **if** $|\mathrm{PNL}| > 1$ **then**
(17) Move all the nodes from PNL to CNL and then set PNL empty
(18) **else**
(19) break;
(20) **end if**
(21) **end while**
(22) **return** The only node left in PNL which is the root of the generated α-filtering tree

Algorithm 4: *GenTree* (L, μ).

$$u \cdot \mathrm{FV}[i] = \max\{u \cdot \mathrm{PL}[1] \cdot \mathrm{FV}[i], u \cdot \mathrm{PL}[2] \cdot \mathrm{FV}[i], \ldots, u \cdot \mathrm{PL}[g] \cdot \mathrm{FV}[i]\},$$
$$\mathrm{Score}\left(V_Q, u \cdot \mathrm{FV}\right) = \sum_{i=0}^{n} u \cdot \mathrm{FV}[i] \times V_Q[i]. \tag{8}$$

Thus, Score (V_Q, $u{\cdot}\mathrm{FV}$) is not less than the relevance scores between any child nodes' filtering vector and the query vector. Then we have, $\forall u' \in u \cdot \mathrm{PL} \longrightarrow u \rhd u'$.

During the search process, for a given Q, if the relevance score between a subtree's root node filtering vector and the corresponding query vector is not higher than the threshold of the candidate result list, then all its child nodes are noncandidates according to Theorem 3. Thus, we can directly ignore this subtree and the search efficiency is improved, which is the pruning criterion of greedy depth first search algorithm. Adopting the idea, we propose a greedy depth first search algorithm shown in Algorithm 5.

In Figure 3, we construct a 3-filtering index tree example to further illustrate multi-keyword ranked search algorithm. The index tree is built any child nodes' filtering vector and the query vector after the leaf nodes are generated from the atom cluster list. The intermediate nodes are generated based on the leaf nodes. We assume that the query vector is $V_Q = (0.5, 0.5, 0, 0)$ and the top-3 ranked documents are interested. When the search starts, the algorithm first visits the left subtree of u_{11}, u_{21}, and u_{31} recursively and finds that u_{31} is a leaf node which has 3 documents. The algorithm puts all the documents into the result list $RL = \{d_1, d_2, d_3\}$, where the relevance scores are 0.3, 0.35, and 0.3, respectively. Then accesses u_{32} is accessed, and the relevance score between its filtering vector and the query vector is 0.2 which is less than 0.3; therefore, RL remains unchanged. After that u_{33} is accessed with the relevance score 0.35, so d_9 and d_{10} are added to RL, replacing d_1 and d_3. Finally, the algorithm searches the subtree rooted by u_{22} and finds no need to search the remaining subtree. The search algorithm is finished. □

6. Effective and Secured Multi-Keyword Ranked Search Scheme

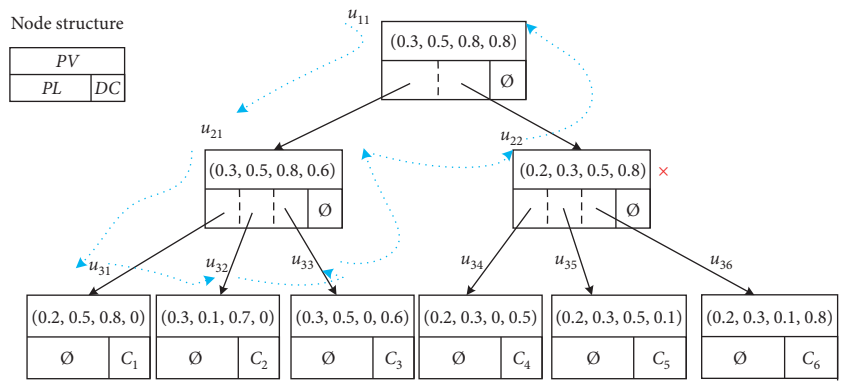
In this section, we construct the secure search scheme by using the secure kNN algorithm [41]. The data owner constructs the index tree with the document set and then uses the secure keys to encrypt the document sets and index tree, respectively. The data user submits search request to the cloud server by using query keywords. The cloud server performs search algorithm on the index tree and returns the search result documents.

6.1. Secure Search Scheme

6.1.1. GenKey ($1^{l(n)}$). The data owner generates secure key $\mathrm{SK} = \{g, S, M_1, M_2\}$, which is used for encrypting the

Input:
The root node of an α-filtering tree, r;
The query vector of Q, V_Q;
The number of requested documents, k;
The minimum of the relevance scores between documents in RL and Q, λ;
The list for storing top-k ranked documents, RL;
Output:
RL;
(1) $u = r$;
(2) **if** u is a leaf node **then**
(3) Add all the documents of u·DC in RL;
(4) **if** $|RL| > k$ **then**
(5) Set the threshold λ equals the minimum of the relevance scores between the candidate documents in RL and V_Q;
(6) Remove the documents from RL, the relevance scores between which and V_Q are smaller than λ;
(7) **end if**
(8) **else**
(9) **if** Score (V_Q, u·FV) $> \lambda$ **then**
(10) **for** each u' in u·PL **do**
(11) *SearchIndex* (u', V_Q, k, λ, RL);
(12) **end for**
(13) **end if**
(14) **end if**

Algorithm 5: *SearchIndex* (r, V_Q, k, λ, RL).



Atom cluster	Document	Document vector
C_1	d_1	(0.2, 0.4, 0.8, 0)
	d_2	(0.2, 0.5, 0.7, 0)
	d_3	(0.2, 0.4, 0.6, 0)
C_2	d_4	(0.3, 0.1, 0.7, 0)
	d_5	(0.3, 0.1, 0.6, 0)
	d_6	(0.3, 0.1, 0.5, 0)
C_3	d_9	(0.3, 0.5, 0, 0.6)
	d_{10}	(0.3, 0.4, 0, 0.6)
C_4	d_{11}	(0.2, 0.2, 0, 0.5)
	d_{12}	(0.1, 0.3, 0, 0.5)
C_5	d_7	(0.1, 0.3, 0.5, 0.1)
	d_8	(0.2, 0.1, 0.5, 0.1)
C_6	d_{13}	(0.2, 0.1, 0.1, 0.8)
	d_{14}	(0.1, 0.3, 0.1, 0.8)
	d_{15}	(0.2, 0.3, 0.1, 0.8)

Figure 3: An example of an α-filtering tree.

documents and index tree. Here, g is the secure symmetric encryption key for document encryption and is only shared with the data user but protected from cloud server. S is a bit vector for vector splitting, and each dimension of S is randomly chosen to be 0 or 1 and the number of 0 and 1 should be nearly equal. M_1 and M_2 are both $n \times n$-dimensional randomly generated invertible matrices.

6.1.2. BuildIndex (D, SK). The data owner first performs index tree construction algorithms discussed in Sections 5.1 and 5.2 to generate the plaintext index tree I on the documents in D. Then, the data owner encrypts the index tree to its encrypted from $\tilde{I}$. Specifically, for each document vector and each node's filtering vector, we use the bit vector S to split them into two vectors. For simplicity, we use V to represent one of these vectors, and the splitting procedures are as follows:

$$\begin{cases} V'[i] + V''[i] = V[i], S[i] = 0, \\ V'[i] = V''[i] = V[i], S[i] = 1. \end{cases} \tag{9}$$

Then the data owner encrypted the split vectors to $\tilde{V} = \{M_1^T V', M_2^T V''\}$. After that, the data owner encrypts the documents in each leaf node's atom cluster by secure key g, and the encrypted index tree $\tilde{I}$ is generated. Finally, the data owner outsources $\tilde{I}$ to the cloud server.

6.1.3. GenTapdoor (Q, SK). The data user generates the query vector V_Q according to the query keywords in Q. Then the secure key SK is adopted to generate the corresponding trapdoor T_Q. The generation of T_Q is similar to the encryption procedures of document vectors. First, V_Q is split into two vectors according to the following equation

$$\begin{cases} V'_Q[i] = V''_Q[i] = V_Q[i], S[i] = 0, \\ V'_Q[i] + V''_Q[i] = V_Q[i], S[i] = 1. \end{cases} \tag{10}$$

Then, the data user encrypted the split vectors into the trapdoor $T_Q = \{M_1^{-1}V'_Q, M_2^{-1}V''_Q\}$. Finally, T_Q is submitted to the cloud server as the search command.

6.1.4. SearchIndex ($\widetilde{I}$, T_Q, k). The cloud server receives the trapdoor T_Q, and performs search algorithm on the secure index tree $\widetilde{I}$. Then, the cloud server returns the encrypted top-k documents result list RL to the data user who decrypts the encrypted documents and the search processing is finished. The special matrix encryption can obtain the inner product of two vectors only with the inner product result of their encryption forms, which is illustrated as follows:

$$\begin{aligned} \text{Score}\left(\widetilde{F_V}, T_Q\right) &= \left\{M_1^T F'_V, M_2^T F''_V\right\} \cdot \left\{M_1^{-1}V'_Q, M_2^{-1}V''_Q\right\} \\ &= \left(M_1^T F'_V\right)^T \cdot M_1^{-1}V'_Q + \left(M_2^T F''_V\right)^T \cdot M_2^{-1}V''_Q \\ &= F'_V \cdot V'_Q + F''_V \cdot V''_Q \\ &= F_V \cdot V_Q \\ &= \text{Score}\left(F_V, V_Q\right). \end{aligned} \tag{11}$$

To protect the Trapdoor unlinkability and keyword privacy under known background model, we should prevent the server from calculating the exact value of the relevance score between the T_Q and F_V which can leak TF distribution information. Thus, we add some phantom terms [11] on the vectors generated in our scheme to disturb the relevance score calculation. But the search accuracy would decrease.

In the enhanced scheme, we generate $(n + n') \times (n + n')$-dimensional secure matrices and also the document vectors will be extended to n dimensions. The extended elements $F_V[n + i]$ are set to a random number β. Similarly, the query vector is also extended to be a $n + n'$ dimensional vector, and the extended elements are random set to 1 or 0. Thus, the relevance score between the query trapdoor and document vector is equal to $F_V \cdot V_Q + \sum \beta_i$, where $V_Q[n + i] = 1$. The randomness of $\sum \beta_i$ can ensure the privacy against the known background model.

6.2. Security Analysis. In this paper, we construct the tree-based secure search scheme same as [11, 42] to achieve searchable encryption, which represents the security of our scheme should be the same as [11, 42]. We give the proof briefly as follows:

(i) Document privacy: we use the traditional symmetric encryption on documents before outsourcing to the cloud server. As long as the secure key is secured against the adversary, the document privacy is protected in our scheme.

(ii) Index and trapdoor privacy: the document vectors and query vectors store the TF and IDF value of the corresponding keywords and encrypted with the secure matrices generated by secure kNN after being randomly split. The secure matrices are both randomly generated invertible matrices. The adversary cannot calculate the secure matrices only with the encrypted vectors. Therefore, the index and trapdoor privacy is protected in our scheme.

(iii) Trapdoor unlinkability: the query trapdoor is randomly split by the split vector S for each search, and the trapdoors are different with same search requests. Thus, the trapdoor unlinkability is guaranteed. But, the cloud server can link the same search requests by inferring the access pattern and the ranked result of the searches. To solve this problem, we can expand the vectors used in our secure scheme by adding phantom dimension to interference the relevance score. With phantom terms, the search results in same requests could be different. However, the search accuracy can be decreased and the balance between the privacy and accuracy is discussed in [11].

(iv) Keyword privacy: the index and trapdoor privacy is protected in our scheme which means keyword privacy is also protected in the known cyphertext model. In the known background model, the relevance score between the documents and the query vector can leak the TF information about the query keywords. If a search request only has one search keywords or one of the search keywords has high TF value, the cloud server can easily infer this keyword by its statistical information about TF distribution of keywords. Similarly, to solve this problem, we add phantom terms to obfuscate the relevance score between the query trapdoor and the document vector. That is to say, the TF-IDF value is variable with different search requests. Thus, the cloud server cannot link the keywords with their TF distribution, and the keyword privacy is enhanced.

7. Performance Analysis

We evaluate the performance of our α-filtering index tree scheme in this section and compare it with Xia et al.'s index tree scheme [11] and Zhu et al.'s HAC-tree [42] under different settings. We use a real-world dataset which has 120000 documents in total and implement our scheme using Java in Windows 10 with an Intel Core i5-6200U @ 2.30 GHz

Table 1: Default parameters.

Parameters	k	μ	$\|Q\|$	α	m
Value	20	10	10	9	120000

CPU, and the default parameter setting is shown in Table 1. k, μ, $|Q|$, α, and m are number of required documents, document threshold in each atom cluster, number of search keywords, number of α, and number of documents, respectively. In the enhanced scheme, we add phantom terms to enhance the security of our scheme. The search accuracy and efficiency of these two schemes are the same without the phantom terms. We only perform evaluation on original scheme for simplicity. The influence of phantom terms is discussed in [11].

7.1. Space Usage Evaluation. In this section, we conduct the space analysis of the different schemes from the aspect of the index tree. We only discuss the index tree space usage; therefore, the search parameters are not changed. The space usage of Xia is the same as Zhu because they both are binary tree with same number of nodes.

7.1.1. Space Usage versus μ. We change the document threshold μ in each cluster to compare the space usage of three schemes. Figures 4(a) and 4(b) shows the index tree space cost when the number of documents is 20000 and 120000, respectively.

The result shows that as the scale of the document set increases, the space usage of index tree is significantly increased. The reason is that more tree nodes are added to the index tree to store more documents. The result also shows that larger threshold can save the space usage of the index tree which will reduce the nodes in the α-filtering tree.

7.1.2. Space Usage versus α. We change α of the α-filtering tree to compare the space usage of three schemes. Figures 5(a) and 5(b) show the index tree space cost when the number of documents is 20000 and 120000, respectively.

The result shows that an appropriate setting of α can largely save the space usage of the α-filtering tree. But when α is too large, the index tree will save space usage with more nodes having the same parent node and tends to be stable.

7.2. Index Building Time Cost Evaluation. In this section, we evaluate the time cost of the index building. We measure the time cost of the *BuildIndex* algorithm of our scheme, which is shown in Table 2, given $m = 20000$. The *BuildIndex* algorithm in our scheme takes hours, while in Xia's scheme, it takes seconds. It should be noted that the key extraction and TF-IDF calculation are the same in all three schemes. And, the tree construction algorithm also has almost the same time cost because the basic structure of the tree is the same. The main difference of the time cost is that our tree uses the clustering algorithm to further improve the search efficiency in search algorithm. The clustering algorithm can consume a lot of time, which leads to worse index building time cost. But, it can be improved by adopting more efficient clustering algorithms such as distributed clustering algorithm. The longer time cost for index building is affordable because it only needs to be performed once while providing more efficient searches.

7.3. Search Time Cost Evaluation. In this section, we evaluate the time cost of the search efficiency of different schemes. Each data point in the figure is at least performed 10 times.

(1) *Time cost versus μ.* Figure 6 indicates that our scheme is better than the existing schemes in the search process. The α-filtering tree can improve the search process by accelerating the process of finding the leaf nodes, shortening the height of the tree, and accessing more nodes by an intermediate node. The k-means cluster can gather similar documents closely in leaf nodes which can fill the candidate result list reasonably. But when u increases, the time cost of our scheme tends to increase simultaneously; the reason is that the number of documents in a leaf node is increased which will slow down the relevance calculation process in the leaf node. The Xia's and Zhu's trees increase largely when the scale of document set increases; the reason is that their schemes are both binary tree in which the height of the tree increases larger than the α-filtering tree in our scheme.

(2) *Time cost versus α.* Figure 7 indicates that the time cost of our scheme is lower than the existing schemes. As mentioned above, appropriate setting of α can improve the performance of our scheme. But when α is too large, the pruning function in an intermediate node will require more calculation and the pruning effect could be worse for there are fewer subtrees to be pruned.

(3) *Time cost versus $|Q|$.* Figure 8 shows that the number of search keywords will slow down the search process of tree based index scheme. But overall, our scheme outperforms other schemes by the contribution of the α-filtering tree.

(4) *Time cost versus k.* Figure 9 shows that under different setting of k, the time cost of our scheme is better than Xia and Zhu. When k increases, the time cost of tree-based schemes increase slightly. The reason is that the pruning function in tree index can save the times of calculation between documents and query vector.

7.4. The setting of α. The experiment shows that different settings of α result in different improvements in our scheme. But it is hard to find an appropriate α for a tree with m nodes. The

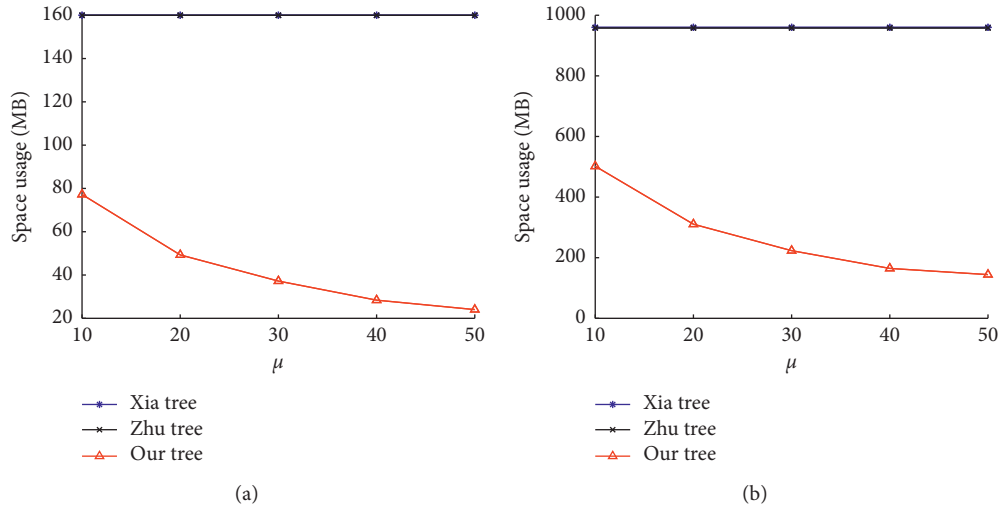


Figure 4: Space usage versus μ. (a) $m = 20000$. (b) $m = 120000$.

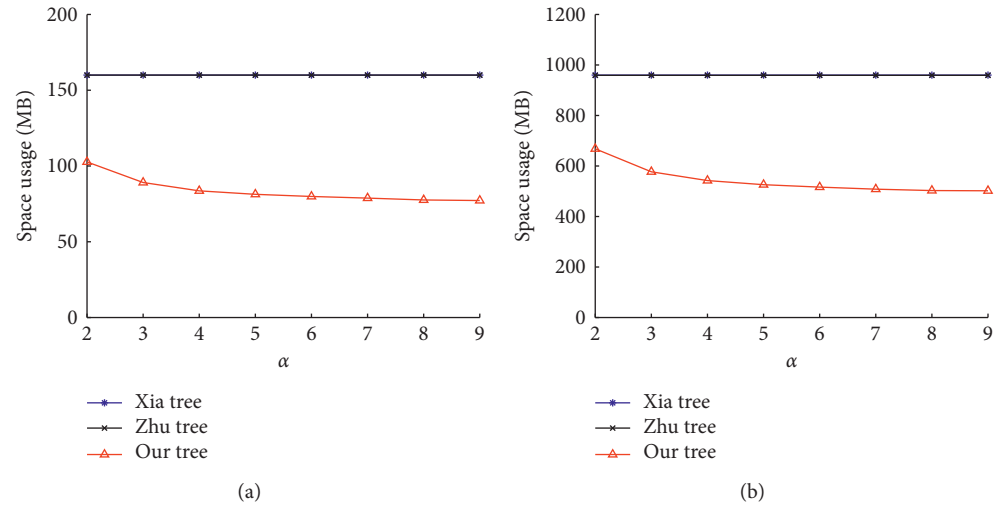


Figure 5: Space usage versus α. (a) $m = 20000$. (b) $m = 120000$.

Table 2: Index building time cost ($m = 20000$).

μ	10	20	30	40	50
Time (min)	294.4	266.9	237.7	231.7	223.45

space usage of α-filtering tree decreases as α increases. However, the search time cost can increase as α increases, and it is worst when $\alpha = m$. When search algorithm iterates every node in the tree, and the filtering vector in the only non-leaf node cannot help to filter the noncandidate nodes. The best α-filtering tree should balance between the width and depth. An α-filtering tree should at least have a depth of three to have the filtering vectors work. The $B+$ tree [43] is a multibranch tree widely adopted for storing index for large data, and arguably degree of a $B+$ tree is usually set to the result of the block size divided by the key size [43] in real circumstances when it stores index for a much larger dataset than that we used in experiment. These settings can help to define an appropriate α. The search time complexity of the α-filtering tree is $O(\log_\alpha m)$, which means that the tree with less depth has better search efficiency. However, the filtering vector in a shorter tree will filter fewer nodes than a tall tree with more filtering vectors, which results in worse search efficiency. Thus, it is hard to define the best setting of α when given different m and it needs further discussion.

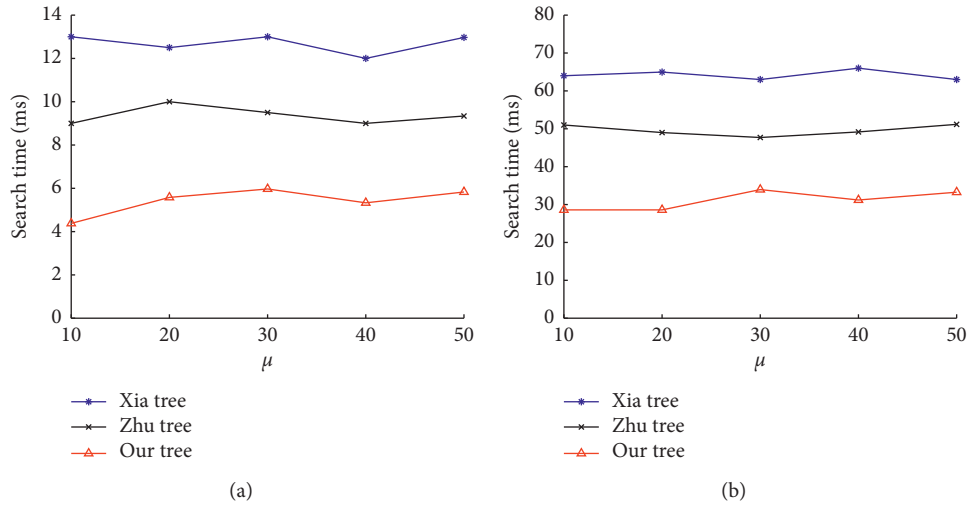


Figure 6: Time cost versus μ. (a) $m = 20000$. (b) $m = 120000$.

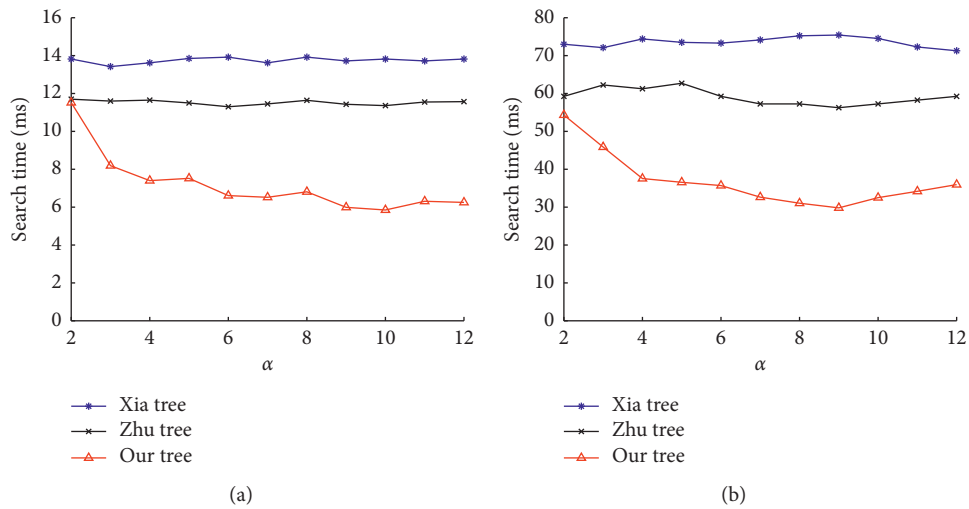


Figure 7: Time cost versus α. (a) $m = 20000$. (b) $m = 120000$.

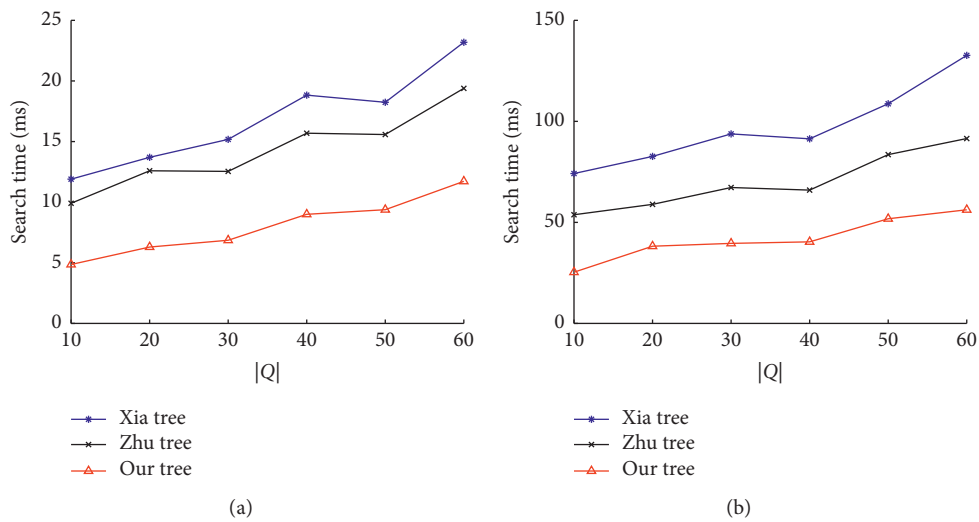


Figure 8: Time cost versus $|Q|$. (a) $m = 20000$. (b) $m = 120000$.

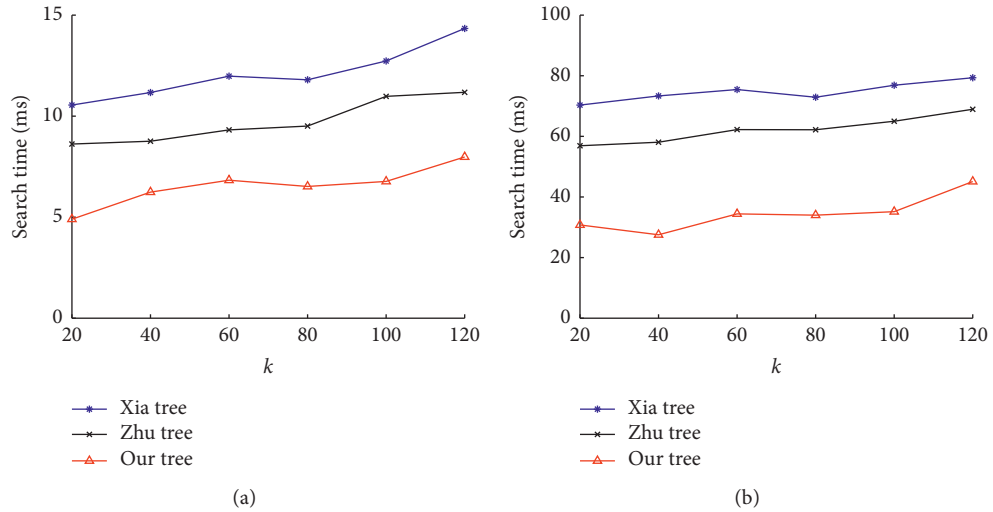


Figure 9: Time cost versus k. (a) $m = 20000$. (b) $m = 120000$.

8. Conclusion

In terms of the efficiency problem of privacy-preserving multi-keyword ranked search, we propose an α-filtering tree index search scheme based on bisecting k-means clusters. The scheme utilizes the characteristics of a multibranch tree, which greatly reduces the spatial complexity of the index tree. At the same time, the idea of clustering is used to store the related documents closely in the index tree, which greatly improves the pruning algorithm on the index tree, thus improving the search efficiency. In contrast, since the index tree nodes are stored in the form of clusters and the clustering of the bisecting k-means requires a large amount of time, the variability of the index tree could be limited. The experiment results on the real-world dataset show that, to a certain extent, our scheme can greatly improve the search efficiency of privacy-preserving multi-keywords ranked search and at the same time guarantee the accuracy of the search results.

Data Availability

The text data used to support the findings of this study are available from the corresponding author upon request.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This research was supported by the National Natural Science Foundation of China under the Grant nos. 61872197, 61972209, 61572263, 61672297, and 61872193; the Postdoctoral Science Foundation of China under the Grant no. 2019M651919; the Natural Science Foundation of Anhui Province under Grant no. 1608085MF127; the Natural Science Foundation of Anhui Province under Grant no. KJ2017A419; and the Natural Research Foundation of Nanjing University of Posts and Telecommunications under Grant no. NY217119.

References

[1] P. Golle, J. Staddon, and B. R. Waters, "Secure conjunctive keyword search over encrypted data," in *Proceedings of Applied Cryptography and Network Security, ACNS 2004*, vol. 31–45, Yellow Mountain, China, June 2004.

[2] Y. H. Hwang and P. J. Lee, "Public key encryption with conjunctive keyword search and its extension to a multi-user system," in *Proceedings of Pairing-Based Cryptography, Pairing 2007*, pp. 2–22, Tokyo, Japan, July 2007.

[3] L. Ballard, S. Kamara, and F. Monrose, "Achieving efficient conjunctive keyword searches over encrypted data," in *Proceedings of the Information and Communications Security, 7th International Conference, ICICS 2005*, vol. 414–426, Beijing, China, December 2005.

[4] D. Boneh and B. Waters, "Conjunctive, subset, and range queries on encrypted data," in *Proceedings of 4th Theory of Cryptography Conference, TCC 2007*, vol. 535–554, Amsterdam, The Netherlands, February 2007.

[5] B. Zhang and F. Zhang, "An efficient public key encryption with conjunctive-subset keywords search," *Journal of Network and Computer Applications*, vol. 34, no. 1, pp. 262–267, 2011.

[6] J. Katz, A. Sahai, and B. Waters, "Predicate encryption supporting disjunctions, polynomial equations, and inner products," in *Proceedings of 27th Annual International Conference on the Theory and Applications of Cryptographic Techniques, EUROCRYPT 2008*, vol. 146–162, Istanbul, Turkey, April 2008.

[7] E. Shen, E. Shi, and B. Waters, "Predicate privacy in encryption systems," in *Proceedings of Theory of Cryptography, 6th Theory of Cryptography Conference, TCC 2009*, pp. 457–473, San Francisco, CA, USA, March 2009.

[8] N. Cao, C. Wang, M. Li, K. Ren, and W. Lou, "Privacy-preserving multi-keyword ranked search over encrypted cloud data," in *Proceedings of the 30th IEEE International Conference on Computer Communications, INFOCOM 2011*, pp. 829–837, Shanghai, China, April 2011.

[9] W. Sun, B. Wang, N. Cao et al., "Privacy-preserving multi-keyword text search in the cloud supporting similarity-based ranking," in *Proceedings of the 8th ACM Symposium on Information, Computer and Communications Security, ASIA CCS '13*, vol. 71–82, Hangzhou, China, May 2013.

[10] C. Örencik, M. Kantarcioglu, and E. Savas, "A practical and secure multi-keyword search method over encrypted cloud data," in *Proceedings of the 2013 IEEE Sixth International Conference on Cloud Computing, CLOUD 2013*, vol. 390–397, Santa Clara, CA, USA, July 2013.

[11] Z. Xia, X. Wang, X. Sun, and Q. Wang, "A secure and dynamic multi-keyword ranked search scheme over encrypted cloud data," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 2, pp. 340–352, 2016.

[12] C. Chen, X. Zhu, P. Shen et al., "An efficient privacy-preserving ranked keyword search method," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 4, pp. 951–963, 2016.

[13] J. Sun, S. Hu, X. Nie, and J. Walker, "Efficient ranked multi-keyword retrieval with privacy protection for multiple data owners in cloud computing," *IEEE Systems Journal*, pp. 1–12, 2019.

[14] D. X. Song, D. A. Wagner, and A. Perrig, "Practical techniques for searches on encrypted data," in *Proceedings of the 2000 IEEE Symposium on Security and Privacy, S&P 2018*, vol. 44–55, Berkeley, California, USA, May 2000.

[15] E. Goh, Secure indexes, IACR Cryptology ePrint Archive, 2003.

[16] Y. Chang and M. Mitzenmacher, "Privacy preserving keyword searches on remote encrypted data," in *Proceedings of the Applied Cryptography and Network Security, Third International Conference, ACNS 2005*, vol. 442–455, New York, NY, USA, June 2005.

[17] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky, "Searchable symmetric encryption: improved definitions and efficient constructions," *Journal of Computer Security*, vol. 19, no. 5, pp. 895–934, 2011.

[18] D. Boneh, G. D. Crescenzo, R. Ostrovsky, and G. Persiano, "Public key encryption with keyword search," in *Proceedings of the International Conference on the Theory and Applications of Cryptographic Techniques, Advances in Cryptology—EUROCRYPT 2004*, vol. 506–522, Interlaken, Switzerland, May 2004.

[19] B. Wang, S. Yu, W. Lou, and Y. T. Hou, "Privacy-preserving multi-keyword fuzzy search over encrypted data in the cloud," in *Proceedings of the 2014 IEEE Conference on Computer Communications, INFOCOM 2014*, pp. 2112–2120, Toronto, Canada, May 2014.

[20] J. Li, Q. Wang, C. Wang, N. Cao, K. Ren, and W. Lou, "Fuzzy keyword search over encrypted data in cloud computing," in *Proceedings of the 29th IEEE International Conference on Computer Communications, INFOCOM 2010*, vol. 441–445, San Diego, CA, USA, 2010.

[21] X. Zhu, Q. Liu, and G. Wang, "A novel verifiable and dynamic fuzzy keyword search scheme over encrypted data in cloud computing," in *Proceedings of the 2016 IEEE International Conference on Trust, Security, and Privacy in Computing and Communications, TrustCom 2016*, pp. 845–851, Tianjin, China, August 2016.

[22] Z. Fu, X. Sun, S. Ji, and G. Xie, "Towards efficient content-aware search over encrypted outsourced data in cloud," in *Proceedings of the 35th Annual IEEE International Conference on Computer Communications, INFOCOM 2016*, vol. 1–9, San Francisco, CA, USA, April 2016.

[23] X. Ge, J. Yu, C. Hu, H. Zhang, and R. Hao, "Enabling efficient verifiable fuzzy keyword search over encrypted data in cloud computing," *IEEE Access*, vol. 6, pp. 45725–45739, 2018.

[24] H. T. Poon and A. Miri, "An efficient conjunctive keyword and phase search scheme for encrypted cloud storage systems," in *Porceedings of the 8th IEEE International Conference on Cloud Computing, CLOUD 2015*, vol. 508–515, New York City, NY, USA, July 2015.

[25] L. Zhang, Y. Zhang, and H. Ma, "Privacy-preserving and dynamic multi-attribute conjunctive keyword search over encrypted cloud data," *IEEE Access*, vol. 6, pp. 34214–34225, 2018.

[26] W. Sun, X. Liu, W. Lou, Y. T. Hou, and H. Li, "Catch you if you lie to me: efficient verifiable conjunctive keyword search over large dynamic encrypted cloud data," in *Proceedings of the 2015 IEEE Conference on Computer Communications, INFOCOM 2015*, pp. 2110–2118, Kowloon, Hong Kong, May 2015.

[27] C.-M. Yu, C.-Y. Chen, and H.-C. Chao, "Privacy-preserving multikeyword similarity search over outsourced cloud data," *IEEE Systems Journal*, vol. 11, no. 2, pp. 385–394, 2017.

[28] C. Wang, K. Ren, S. Yu, and K. M. R. Urs, "Achieving usable and privacy-assured similarity search over outsourced cloud data," in *Proceedings of the IEEE Conference on Computer Communications, INFOCOM 2012*, vol. 451–459, Orlando, FL, USA, March 2012.

[29] Z. Xia, Y. Zhu, X. Sun, and L. Chen, "Secure semantic expansion based search over encrypted cloud data supporting similarity ranking," *Journal of Cloud Computing*, vol. 3, no. 1, p. 8, 2014.

[30] W. Sun, B. Wang, N. Cao et al., "Verifiable privacy-preserving multi-keyword text search in the cloud supporting similarity-based ranking," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 11, pp. 3025–3035, 2014.

[31] P. van Liesdonk, S. Sedghi, J. Doumen, P. H. Hartel, and W. Jonker, "Computationally efficient searchable symmetric encryption," in *Proceedings of the Secure Data Management, 7th VLDB Workshop, SDM 2010*, vol. 87–100, Singapore, September 2010.

[32] X. Ge, J. Yu, H. Zhang et al., "Towards achieving keyword search over dynamic encrypted cloud data with symmetric-key based verification," *IEEE Transactions on Dependable and Secure Computing*, pp. 1–16, 2019.

[33] S. Kamara, C. Papamanthou, and T. Roeder, "Dynamic searchable symmetric encryption," in *Proceedings of the ACM Conference on Computer and Communications Security, CCS'12*, pp. 965–976, Raleigh, NC, USA, October 2012.

[34] D. Cash, J. Jaeger, S. Jarecki, and G. Tsudik, "Dynamic searchable encryption in very-large databases: data structures and implementation," in *Proceedings of the 21st Annual Network and Distributed System Security Symposium, NDSS 2014*, San Diego, CA, USA, February 2014.

[35] F. Hahn and F. Kerschbaum, "Searchable encryption with secure and efficient updates," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, CCS'14*, vol. 310–320, Scottsdale, AZ, USA, November 2014.

[36] L. Chen and Z. Chen, "Practical, dynamic and efficient integrity verification for symmetric searchable encryption," in *Proceedings of the Cryptology and Network Security—18th International Conference, CANS 2019*, vol. 163–183, Fuzhou, China, October 2019.

[37] X. Jiang, J. Yu, J. Yan, and R. Hao, "Enabling efficient and verifiable multi-keyword ranked search over encrypted cloud data," *Information Sciences*, vol. 403-404, pp. 22–41, 2017.

[38] Q. Wang, M. He, M. Du, S. S. M. Chow, R. W. F. Lai, and Q. Zou, "Searchable encryption over feature-rich data," *IEEE Transactions on Dependable and Secure Computing*, vol. 15, no. 3, pp. 496–510, 2018.

[39] S. Hu, L. Y. Zhang, Q. Wang, Z. Qin, and C. Wang, "Towards private and scalable cross-media retrieval," *IEEE Transactions on Dependable and Secure Computing*, pp. 1–15, 2019.

[40] C. Wang, N. Cao, K. Ren, and W. Lou, "Enabling secure and efficient ranked keyword search over outsourced cloud data," *IEEE Transactions on Parallel and Distributed Systems*, vol. 23, no. 8, pp. 1467–1479, 2012.

[41] W. K. Wong, D. W. Cheung, B. Kao, and N. Mamoulis, "Secure kNN computation on encrypted databases," in *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2009*, vol. 139–152, Providence, RI, USA, June 2009.

[42] X. Zhu, H. Dai, X. Yi, G. Yang, and X. Li, "MUSE: an efficient and accurate verifiable privacy-preserving multikeyword text search over encrypted cloud data," *Security and Communication Networks*, vol. 2017, Article ID 1923476, 17 pages, 2017.

[43] Wikipedia contributors, B+ tree—wikipedia, the free encyclopedia," October 2019, https://en.wikipedia.org/w/index.php?title=B%2B_tree&oldid=920193380.