Hindawi
Security and Communication Networks
Volume 2017, Article ID 1923476, 17 pages
https://doi.org/10.1155/2017/1923476

Research Article

MUSE: An Efficient and Accurate Verifiable Privacy-Preserving Multikeyword Text Search over Encrypted Cloud Data

Zhu Xiangyang,[1] Dai Hua,[1,2] Yi Xun,[3] Yang Geng,[1,2] and Li Xiao[1]

[1]*College of Computer Science and Technology, Nanjing University of Posts and Telecommunications, Nanjing 200013, China*
[2]*Jiangsu Key Laboratory of Big Data Security and Intelligent Processing, Nanjing 210013, China*
[3]*School of Computer Science and IT, RMIT University, Melbourne, VIC 3001, Australia*

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

Received 9 February 2017; Accepted 22 May 2017; Published 11 July 2017

Academic Editor: Xiangyang Luo

With the development of cloud computing, services outsourcing in clouds has become a popular business model. However, due to the fact that data storage and computing are completely outsourced to the cloud service provider, sensitive data of data owners is exposed, which could bring serious privacy disclosure. In addition, some unexpected events, such as software bugs and hardware failure, could cause incomplete or incorrect results returned from clouds. In this paper, we propose an efficient and accurate verifiable privacy-preserving multikeyword text search over encrypted cloud data based on hierarchical agglomerative clustering, which is named MUSE. In order to improve the efficiency of text searching, we proposed a novel index structure, HAC-tree, which is based on a hierarchical agglomerative clustering method and tends to gather the high-relevance documents in clusters. Based on the HAC-tree, a noncandidate pruning depth-first search algorithm is proposed, which can filter the unqualified subtrees and thus accelerate the search process. The secure inner product algorithm is used to encrypted the HAC-tree index and the query vector. Meanwhile, a completeness verification algorithm is given to verify search results. Experiment results demonstrate that the proposed method outperforms the existing works, DMRS and MRSE-HCI, in efficiency and accuracy, respectively.

1. Introduction

IT resources, such as computing and storage, are treated as the on-demand services in clouds nowadays, which is shown as "X as a Service." In order to reduce the cost of data management and storage, data owners (DOs) who have a large amount of data usually choose to outsource their data to clouds. However, DOs cannot directly control their data placed in remote cloud servers, which may cause concerns about their outsourced data being illegally acquired or abused by the cloud service providers (CSPs), especially for the privacy-sensitive data, such as medical records, government documents, and emails. Although many CSPs claim that their cloud services have several security countermeasures, such as access control, firewall and intrusion detection, doubts about the security, and privacy of outsourced data that are still the main obstructions to the wider development of cloud computing [1].

A general approach to protecting data privacy is to encrypt the data before outsourcing [2]. However, this will make a significant difficulty and cost in terms of data management and utilization. In the field of information retrieval (IR), the existing retrieval technique based on multikeyword is mainly for the plaintext data and cannot be directly applied to the encrypted data. It is obviously unrealistic and wasteful to download all encrypted data from clouds to the local for decryption. In addition, due to hardware/software failures, storage device failure, and so forth, the search results may contain corrupted or incorrect data. If users cannot verify the completeness and correctness of search results, the upper-level decisions based on the search results may be misleading. Therefore, it is a challenge to research and give a searchable encryption scheme that supports verifiable privacy-preserving multikeyword text search over encrypted cloud data, which has become one of the hot issues in cloud computing recently [3–8].

In order to deal with the above problems, some encrypted data search methods [9–12] are proposed which utilize kinds of cryptographic techniques, such as homomorphic encryption and public-key cryptography. They are proved secure in text searching, but they usually need massive mathematical operations and cause high computational overhead. Hence, these methods do not adapt to the cloud computing scenario where data storage is very large and online data processing is the basic requirement. Besides, the relationship between documents is not taken into account during the search process, such as the category which describes the classification relationship of documents. In the design of encrypted documents search scheme, if we consider this type of documents relationship, it could improve the search efficiency and accuracy. However, the category relationship has been concealed by the blind encryption in the traditional methods. Therefore, it is desirable and helpful to maintain and utilize the category relationship to perform efficient and accurate text search for encrypted outsourced documents.

In this paper, we propose an efficient and accurate verifiable privacy-preserving multikeyword text search over encrypted cloud data (MUSE), which is based on hierarchical agglomerative clustering tree index (HAC-tree). We use the TF-IDF model and vector space model to represent every document and the interested keywords of queries as vectors, which means that each of them is considered as a point in a high-dimensional space. The secure inner product computation, adapted in the secure kNN [13], is used for measuring the relevance score between two documents or a document and a query. Based on the hierarchical agglomerative clustering, we propose a novel index structure, HAC-tree, which is constructed from the bottom leaf nodes to the upper root node. Firstly, documents are initialed as leaf nodes. Then the internal nodes are generated by clustering pairs of child nodes according to the sequence of their relevance scores level by level until the root node. Each internal node is with a pruning vector which is the extract maximum vector of its child nodes. On the basis of HAC-tree, we propose a noncandidate pruning depth-first search algorithm (NCP-DFS) for searching top-k relevance score documents recursively. The pruning vectors of internal nodes are used to filter the noncandidate subtrees which impossibly contain search results; thus the search space is narrowed and the search process speeds up while the result accuracy is not reduced. To verify the completeness and correctness of the search results, digests are generated for documents which are stored in leaf nodes of the HAC-tree. Upon returning the search results, a verification object (VO) is constructed for the result documents which is returned to data user (DU) along with the results. When DU receives the returned data from CS, a VO reconstruction procedure is performed to verify the completeness and correctness of the search results.

Our contributions of this paper are summarized as follows:

(1) According to the basic idea of hierarchical agglomerative clustering, we propose a novel index structure HAC-tree and the corresponding bottom-up construction algorithm. The HAC-tree is the lowest binary tree if the leaf nodes number is fixed. Moreover, documents with higher relevance score between them (which means those documents may belong to similar categories) are always clustered.

(2) On the basis of HAC-tree, we propose a noncandidate pruning depth-first search algorithm (NCP-DFS) for multikeyword text search. It can prune the subtrees that surely do not contain any search results; thus the search efficiency will be improved.

(3) We define digests for documents and propose a result verification algorithm to check whether the search results are complete and correct. This algorithm can detect the damaged or incomplete results caused by hardware of software faults.

(4) Based on the above methods, we propose two verifiable privacy-preserving multikeyword text search schemes over encrypted cloud data, named BMUSE and EMUSE, respectively. BMUSE is a basic scheme which can resist known ciphertext threat, while EMUSE is an enhanced scheme which can resist known background threat.

(5) We compare the proposed method to DMRS [7] and MRSE-HCI [8] on a real dataset. Our results show that (i) the proposed method outperforms DMRS in terms of efficiency without losing accuracy; and (ii) the proposed method consistently gives more accurate results than MRSE-HCI while it may be less efficient than MRSE-HCI, depending on the choices of parameter values in MRSE-HCI.

The paper is organized as follows. Section 2 describes the related work. Section 3 gives the main notations and necessary preliminaries. Section 4 gives model and problem statement. Section 5 presents the index structure HAC-tree and its construction algorithm. On the basis of HAC-tree, the plaintext multikeyword search algorithm and search results verification are given. Section 6 presents the basic and enhanced schemes of the secure multikeyword text search. In Section 7, we analyze the security of the schemes. Section 8 carries on the experiment, comparing with the existing schemes in terms of the search results accuracy and the search efficiency. Section 9 concludes this paper.

2. Related Work

Searchable encryption (SE) allows data owner to encrypt their own documents before uploading to the cloud server. In recent years, searchable encryption has drawn a wide range of attention, for example, [14–25].

(i) Single Keyword Searchable Encryption. Song et al. [26] first propose symmetric searchable encryption based on pseudorandom function and symmetric encryption mechanism and proof the security rigorously. Goh [14] gives security definitions for the formalization of the security requirements of searchable symmetric encryption schemes. Subsequently, many improvement and novel methods are proposed [27–29]. Boneh et al. [10] propose a public-key ciphertext search

algorithm named PEKS based on bilinear mapping and IBE encryption. In this algorithm, encrypted data by the public key can be authenticated by the gateway and sent to the corresponding user, but the real content will not be revealed. However, none of the above works ranks the results. The cloud server has to return all the results to meet the query request, resulting in unnecessary bandwidth and processing power overhead. In order to return only the most relevant search results, Wang et al. [30] present keyword ranked search over encrypted data based on TF × IDF and Order-Preserving Symmetric Encryption (OPSE). However, the above works only focus on single keyword search and cannot be applied to the scenario of multikeyword search which is the main concern of this paper.

(ii) Multikeyword Searchable Encryption. In the typical case of search over encrypted data in cloud computing, a single keyword search cannot express the data user search intention sufficiently. The cloud server will inevitably return an excessive number of matches, where most will probably be irrelevant for the user. Multikeyword search allows data users to characterize their own requests from multiple perspectives, ensuring that the search results are the most relevant documents with the query. Bilinear pairing-based solutions are presented in [31–33]. The results in bilinear pairing-based solutions are free from false positives and false negatives caused by hashing. However, computation costs of pairing-based solutions are prohibitively high both on the server and on the user side. Pang et al. [34] propose a secure search scheme based on vector space model. Without the security analysis for frequency information and practical search performance, it is unclear whether there is keyword privacy disclosure or not. Besides, the practical search performance is absent from the demonstration of their experiment. Cao et al. [3] define and solve the challenging problem of privacy-preserving multikeyword ranked search over encrypted cloud data (MRSE), which adopt the similarity measure of "coordinate matching" to capture the relevance of data documents to the search query and ignore frequency information, leading to low accuracy in results. Meanwhile, MRSE needs a huge computational overhead. Sun et al. [4] use MDB-tree as index structure to improve the efficiency of MRSE, named MTS. Each level of the MDB-tree represents a subvector instead of an attribute domain in the database scenario, which leads to a decrease in accuracy. The index vector clustering further degrades the retrieval accuracy. Xia et al. [7] present a secure and dynamic multikeyword ranked search scheme (DMRS), which conduct a tree-based index structure to ensure accurate relevance score calculation between the encrypted document vector and the query vector. DMRS is significantly superior to MTS in accuracy. However, owing to neglecting the relationship of documents, this scheme still brings a lot of calculation cost. Therefore, there is still much room for improvement in search efficiency. In this paper, we focus on how to utilize the relationship of documents to improve the search efficiency.

Obviously, when constructing the index, if the similarity of documents can be classified and make their access paths as close as possible, it will make a huge contribution to the search efficiency through multikeyword of interest. Chen et al. [8] use the idea to propose a multikeyword ranked search over encrypted data based on hierarchical clustering index (MRSE-HCI). This method uses K-means to cluster the documents based on the minimum relevance threshold and the maximum size of subcluster. Searching in the most relevant subcluster can achieve a linear computational complexity against an exponential size increase of document collection. Nevertheless, the significant improvement in efficiency is at the expense of accuracy, which reduces the accuracy and cannot fulfill user expectation well. In this paper, we not only improve the top-k search efficiency but also ensure the accuracy of search results. Although the search efficiency of our work may be lower than MRSE-HCI in some cases depending on the choices of parameter values in MRSE-HCI, the search result accuracy is much higher than MRSE-HCI.

In addition, Sun et al. [35] use Merkle hash tree and cryptographic signature to build a verifiable MDB-tree. Chen et al. [8] design a minimum hash subtree as a verifiable structure. However, their works need to transmit a lot of verification object. Wan and Deng [36] propose the adapted homomorphic MAC technique and random challenge technique with ordering for verifying top-k search results. However, the method requires a linear search of all documents and have poor efficiency performance. Hence, a proper mechanism should be adopted which really reduces the transmission cost of the verification object.

3. Notations and Preliminaries

3.1. Notations. For the sake of clarity, we firstly introduce the main notations used in this paper:

(i) FS: a plaintext document collection, FS $= \{F_1, F_2, \ldots, F_m\}$

(ii) E: an encrypted document collection, $E = \{E_1, E_2, \ldots, E_m\}$

(iii) W: a keyword dictionary including n keywords, $W = \{w_1, w_2, \ldots, w_n\}$

(iv) I: a binary tree index generated from FS (each leaf node is associated with a document in FS)

(v) $\widetilde{I}$: a searchable encrypted index which is generated from I

(vi) I_p: a set of plaintext index, $I_p \subseteq I$

(vii) $\widetilde{I}_p$: the encrypted form of I_p

(viii) u: a node of the index tree I

(ix) C_u: a cluster whose items are documents represented by leaf nodes of the subtree with node u as its root

(x) V_d: a vector of document F_d

(xi) $\widetilde{V}_d$: the encrypted form of V_d

(xii) S_q: a query consisting of a set of the interested keywords, $S_q = \{w_1, w_2, \ldots, w_t\} \subseteq W$

(xiii) Q: the vector of S_q

(xiv) $\widetilde{Q}$: the encrypted form of Q

(xv) R: a list storing the search results

(xvi) $\mathrm{Enc}(k_f, *)$, $\mathrm{Dec}(k_f, *)$: symmetric encryption and decryption functions where k_f is a private key

3.2. Preliminaries

Vector Space Model. Vector space model along with TF-IDF algorithm is very popular in the information retrieval area, which is also widely used in secure multikeyword search [3, 4, 7, 8, 16, 18]. We adopt the classic definitions of term frequency (TF) and inverse document frequency (IDF). The former refers to the number of times a given keyword or term exists in documents while the latter is calculated through dividing the total number of documents in the collection by the number of documents having the keyword. Under the vector space model, each document F_d is denoted as a n-dimensional vector V_d. Any element $V_d[i]$ in V_d stores the normalized TF value of the keyword w_i whose calculation formula is shown in (1). Similar to documents, the interested keywords S_q are also denoted as a n-dimensional vector, named the query vector Q, whose element $Q[i]$ stores the normalized IDF value of the keyword w_i in S_q. The calculation formula of $Q[i]$ is shown in (2). Obviously, the lengths of the document vector and query vector are equal to the capacity of the keywords dictionary and each element of them is nonnegative.

$$V_d[i] = \frac{\mathrm{TF}_{d,w_i}}{\sqrt{\sum_{w_i \in F_d} \left(\mathrm{TF}_{d,w_i}\right)^2}}, \tag{1}$$

$$Q[i] = \frac{\mathrm{IDF}_{w_i}}{\sqrt{\sum_{w_i \in S_q} \left(\mathrm{IDF}_{w_i}\right)^2}}, \tag{2}$$

where TF_{d,w_i} is the TF value of w_i in F_d, $\mathrm{TF}_{d,w_i} = (1+\ln f_{d,w_i})/|f_d|$, IDF_{w_i} is IDF value of w_i in FS, $\mathrm{IDF}_{w_i} = \ln(1 + m/f_{w_i})$, f_{d,w_i} is the frequency of w_i that appears in F_d, and f_{w_i} is the number of documents containing w_i in FS.

Relevance Score Measurement. In this paper, we use the same measurements in [8] to quantify the relevance score between a pair of documents and between a document and a query (which is represented by the interested keywords of DU). It is also used to quantify the relevance score between a pair of cluster centers and between a cluster center and a query. The calculation of the above relevance score can be unified as the inner product of two vectors which is shown in

$$\mathrm{score}(p, q) = p \cdot q = \sum_{i=1}^{n} p[i] \times q[i]. \tag{3}$$

Secure Inner Product Operation. To achieve the goal of privacy preserving, we adopt the secure inner product operation which is proposed in [13]. The operation is able to calculate the inner product of two vectors without knowing their plaintext value. Its basic idea is as follows. Assume that p and q are two n-dimensional vectors and M is a random $n \times n$ invertible matrix that is treated as a secure key. The encrypted forms of p and q are denoted as $\widetilde{p}$ and $\widetilde{q}$, respectively, where $\widetilde{p} = M^T p$, $\widetilde{q} = M^{-1} q$. The inner product of $\widetilde{p}$ and $\widetilde{q}$ is calculated as (4) which indicates that $\widetilde{p} \cdot \widetilde{q} = p \cdot q$. Note that we can get the inner product of two vectors without knowing the plaintext.

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

The space vector model, inner product of two vectors, and secure inner product operation are widely used in the existing works [3, 4, 7, 8, 17, 18, 35]. In this paper, we will use them to design the secure multikeyword text search schemes over encrypted cloud data.

4. Model and Problem Statement

4.1. System Model. The system considered in this paper is the same as [3–5, 7, 8, 16–18] which consists of three entities: the data owner (DO), the data user (DU), and the cloud server (CS). As shown in Figure 1, their collaboration is as follows.

(1) DO Owns the Sensitive Data. To protect the privacy of the data, DO encrypts FS into ciphertext E. In order to make E searchable in CS, DO constructs a secure index $\widetilde{I}$ on FS. Then DO outsources E and $\widetilde{I}$ into CS. Besides, DO can grant the authorizations of accessing their outsourced data to DU.

(2) CS Provides Services, Such as Data Storage and Management. On one hand, CS is in charge of storing the outsourced data from DOs. On the other hand, upon receiving a search trapdoor T_Q from DU, CS performs the secure search by $\widetilde{I}$ and returns the qualified encrypted documents as search results to DU.

(3) DU Is the User Authorized by DO Who Can Legally Search the Data Stored in CS. When DU starts a multikeyword search, its interested keywords are transformed into a corresponding trapdoor T_Q according to the authorization from DO. Then T_Q is submitted to CS as the search command. After DU receives search results from CS, it decrypts the ciphertext and verifies the completeness and correctness of the result.

4.2. Search Model. Given a set of t interested keywords $S_q = \{w_1, w_2, \ldots, w_t\}$, a multikeyword text search is the process of calculating relevance scores between all documents in FS and S_q and returning the top-k results from the highest score. Formally, we define the multikeyword search as follows:

$$\mathrm{Query} = \left(\mathrm{FS}, S_q, k\right), \tag{5}$$

where k is the quantity of requested documents and $k \ll m$ generally.

Assuming that S_q is represented by a query vector Q, each document $F_i \in \mathrm{FS}$ is represented by a document vector V_i, and the relevance score between V_i and Q is calculated according to (3). Then, the search results R of $\mathrm{Query} = (\mathrm{FS}, S_q, k)$ satisfies

$$\begin{aligned} &|R| = k \wedge \forall F_i, F_j \\ &\left(F_i \in R \wedge F_j \in (\mathrm{FS} - R)\right) \longrightarrow \mathrm{score}(V_i, Q) \geq \mathrm{score}\left(V_j, Q\right). \end{aligned} \tag{6}$$

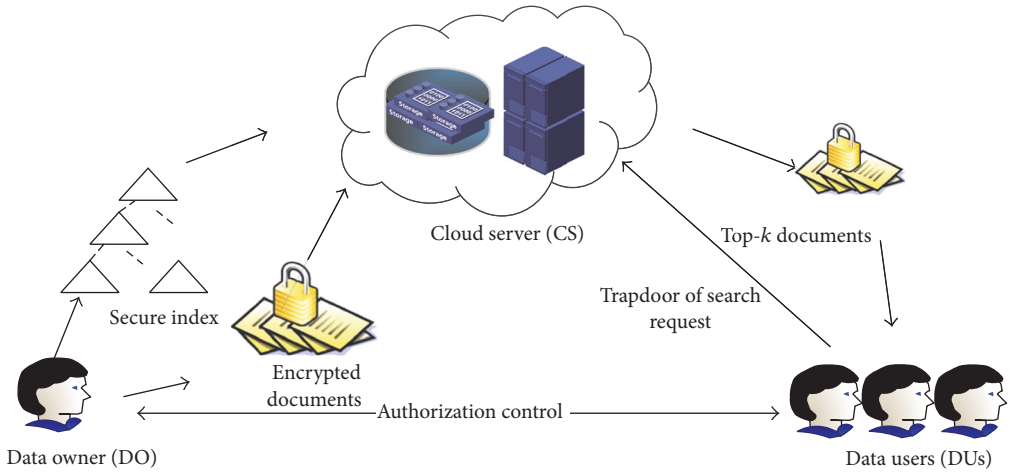


Figure 1: System model.

4.3. Threat Model. We consider the same threat model as [3–5, 7, 8, 16–18], which assumes that DO and DU are trusted, but CS is considered as "honest-but-curious" model. That is to say, CS honestly implements protocols and returns search results, but it is curious about inferring and analyzing the plaintext outsourced data, searchable index, and messages that are received during protocol executions. We assume that CS knows not only ciphertext but also the encryption and decryption algorithms, but it has no idea of keys. According to the background information that CS knows, we adopted two threat models as follows.

Known Ciphertext Threat Model. In this model, CS not only observes the ciphertext, including the encrypted documents E, the secure index $\widetilde{I}$, and the search trapdoor T_Q, but also knows a small part of plaintext index I_p and the corresponding encrypted index $\widetilde{I}_p$. The threat model here is more aggressive than the known ciphertext model adopted in [3, 7, 8] and is the same as Level 3 adopted in [18].

Known Background Threat Model. In this stronger model, CS is supposed to possess more knowledge than the known ciphertext model, such as the term frequency statistics of document collection. This statistical information records how many documents are there for each term frequency of a specific keyword in FS, which can be utilized by CS to apply TF statistical attacks and hence deduce or even identify certain keywords through analyzing histogram and value range of the corresponding frequency distributions [7].

In addition, in some unexpected events, such as software bugs and internal/external attacks, CS may behave beyond the "honest-but-curious" model and return incorrect or incomplete search results. Thus, enabling search results verification mechanism that can detect useless results is also significant in terms of practicality and worth further investigation.

4.4. Problem Statement. Given a search request Query = (FS, S_q, k) and R is the corresponding search results returned from CS. We focus on the multikeyword text search schemes over encrypted cloud data, whose goal is to achieve the following functions and security guarantees:

(i) Multikeyword search: CS is able to return R which includes k most relevant documents with the query.

(ii) Search efficiency and accuracy: the comprehensive performance of search efficiency and accuracy should be ensured which means that the search process should be efficient while the accuracy of search results should not be reduced.

(iii) Privacy preserving: in the search process, the schemes should not reveal the confidentiality of documents, index, query keywords, and trapdoor unlinkability [3, 7, 8].

(iv) Completeness verification: DO can verify the completeness and correctness of search results to make sure that the result documents are right and useful.

4.5. MUSE Framework. To achieve the multikeyword search scheme over encrypted cloud data (MUSE), we give the framework of MUSE shown in Figure 2. It consists of five modules, such as *GenKey*, *BuildIndex*, *GenTrapdoor*, *DoSearch*, and *Verify*, whose functions are described briefly as follows.

GenKey. DO generates the secure key SK.

BuildIndex. DO constructs the plaintext index tree I and then uses SK to encrypt I into the encrypted form $\widetilde{I}$.

GenTrapdoor. DU transforms its interested query keywords S_q into the trapdoor T_Q. Then DU sends T_Q to CS.

DoSearch. CS performs the secure multikeyword text search to obtain search results and then return them to DU.

Verify. DU receives the search results from CS and verifies their completeness and correctness.

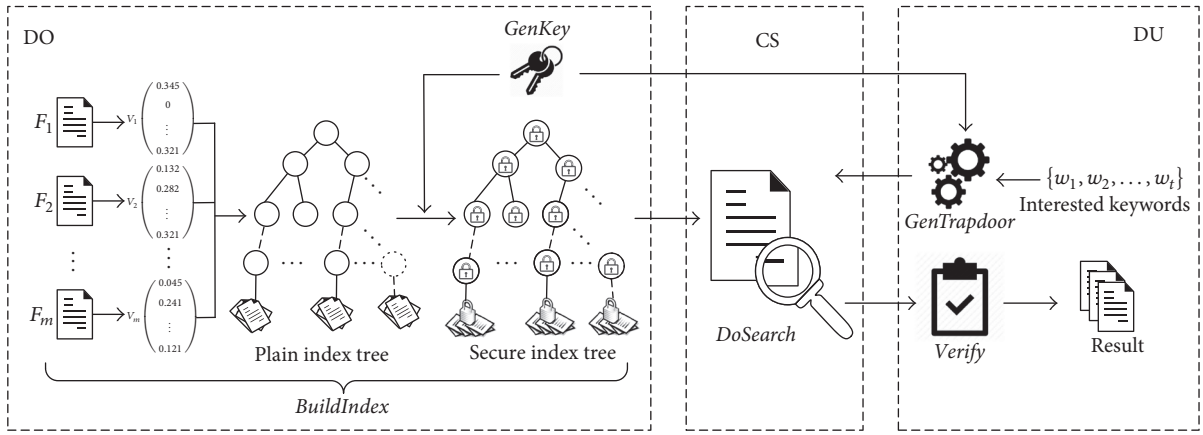


Figure 2: Framework of the secure multikeyword search.

5. Index Design and Algorithms

According to the existing works, we know that the index is the key to realizing the secure multikeyword text search over encrypted cloud data, and its structural characteristics directly determine the search efficiency and accuracy of search schemes. It is reasonable that if documents in the same category are settled in the nearby access paths in an index, the search efficiency could be improved. Therefore, we adopt a hierarchical agglomerative clustering technology to design a novel index structure HAC-tree and give the construction algorithm. On the basis of HAC-tree, an efficient and accurate multikeyword search algorithm and search results verification algorithm are proposed. It is noticeable that the index in this section is unencrypted.

5.1. Hierarchical Agglomerative Clustering. The hierarchical agglomerative clustering (HAC) is one of the important and commonly used clustering methods in data mining area. In the hierarchical agglomerative clustering procedure, the lower-level cluster pairs are merged into higher-level clusters according to the similarity of clusters until the root cluster is formed. By this way, a binary clustering tree is constructed from the bottom leaf nodes to the upper root node. A leaf node of a tree represents an initial cluster containing only one item while an internal node represents a cluster consisting of the items stored in the leaf nodes of the subtree whose root is the internal node.

Here, we use the relevance score to represent the similarity between a pair of clusters. The relevance score of a pair of clusters C_i and C_j is score(V_i, V_j) where V_i and V_j are the cluster center vectors of C_i and C_j, respectively. The cluster center vector computation is shown in the following definition.

Definition 1. One assumes that $C_i = \{u_1, u_2, \ldots, u_m\}$ is a cluster where $u_i \in C_i$ is an item and its corresponding vector is V_i. The cluster center vector of C_i is denoted as V_0, then we have

$$V_0[j] = \frac{\sum_{t=1}^{m} V_t[j]}{m}. \quad (7)$$

To describe the HAC method clearly, we give an example as shown in Figure 3 where items are documents in the collection FS $= \{F_1, F_2, \ldots, F_8\}$. The relevance scores between these documents are mapped to point-to-point distances in the 2-dimensional space where a point represents a document as shown in Figure 3(a). If the distance of a pair of documents is closer than the other pair, the relevance score of the former is higher than the latter. Figure 3(b) shows the clustering procedure and the corresponding generated clustering tree. At the beginning, documents in FS are initialized to 8 clusters that are corresponded to 8 leaf nodes in the clustering tree, that is, $\{F_1\}, \{F_2\}, \ldots, \{F_8\}$. Then, round 1 of clustering starts which takes the 8 initial clusters as the current operated clusters. Their nearest pairs are merged into 4 new clusters which are corresponded to the parent nodes of the leaf nodes in the clustering tree, that is, $\{F_4, F_8\}$, $\{F_3, F_7\}$, $\{F_2, F_6\}$, $\{F_1, F_5\}$. After that, the new generated clusters are taken as the current operated clusters and round 2 of clustering starts similarly, so does round 3. The clustering process ends when the new cluster containing all documents of FS is generated which means that the root of the clustering tree is generated. It is noticeable that if the number of current operated clusters is odd, then there must be one of the current operated clusters left for the next round clustering.

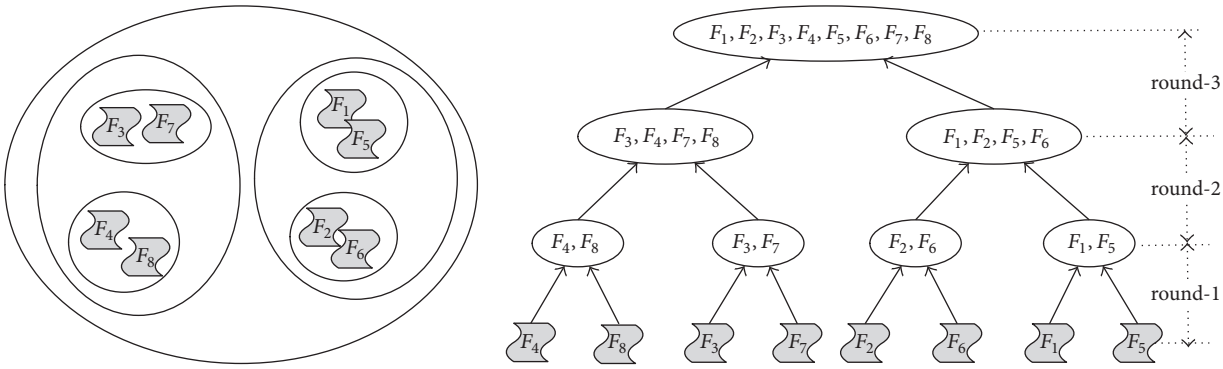
According to the above example as shown in Figure 3, a binary tree is constructed which has the following lemma.

Lemma 2. *Let the tree constructed by the HAC method be T and one assumes there are m leaf nodes in T. Thus, we have the following:*

(1) *The height of T is* $\lceil \log_2 m \rceil + 1$.

(2) *The height of any binary tree with m leaf nodes is not lower than* $\lceil \log_2 m \rceil + 1$.

Proof. (1) According to the HAC procedure, there are $m, \lceil m/2 \rceil, \lceil m/2^2 \rceil, \ldots, \lceil m/2^{x-1} \rceil$ current operated nodes (clusters) in round 1, round 2, round 3, ..., and round x processing, respectively. The construction of T is finished when there is only one current operated node left which is indeed the root node; that is, $\lceil m/2^{x+1} \rceil = 1$. Thus, we have



(a) The relevance scores between documents described by the distances between them in the 2-dimensional space

(b) The hierarchical agglomerative clustering procedure and the corresponding constructed tree

Figure 3: An example of hierarchical agglomerative clustering.

$x = \lceil \log_2 m \rceil + 1$. Obviously, the height of T is equal to the number of clustering rounds. Therefore, we have that the height of T is $\lceil \log_2 m \rceil + 1$ and Lemma 2(1) holds.

(2) Then, we give the proof by contradiction for Lemma 2(2). We assume that there is a binary tree T', having the same number of leaf nodes with T, the height of which is smaller than $\lceil \log_2 m \rceil + 1$. Let us assume that the height of T' is $\lceil \log_2 m \rceil$. We all know that a binary tree, whose height is h, has at most $2^h - 1$ nodes and the 0-degree nodes (which are leaf nodes) are 1 more than the 2-degree nodes (which are internal nodes having both left and right child nodes). Hence, we can get two observations: (1) T' has at most $2^{\lceil \log_2 m \rceil} - 1$ nodes. Because $2^{\lceil \log_2 m \rceil} - 1 < 2^{1+\log_2 m} - 1$, then $2^{\lceil \log_2 m \rceil} - 1 < 2m - 1$. It means that T' has less than $2m - 1$ nodes. (2) According to the given assumption, we known that T' has m 0-degree nodes, thus it has $m-1$ 2-degree nodes. A binary tree only has three kinds of nodes, 2-degree, 1-degree, and 0-degree nodes. Because the number of 1-degree nodes is at least 0, T' has no less than $2m - 1$ nodes. Obviously, a contradiction appears between the above two observations. Therefore, the height of any binary tree with m leaf nodes is not lower than $\lceil \log_2 m \rceil + 1$. Lemma 2(2) holds. □

Lemma 2 indicates that the tree constructed by the HAC method is the lowest of the binary trees which have the same number of leaf nodes. Thus, we can have that the time complexity from the root to any leaf node is no more than $\lceil \log_2 m \rceil + 1$.

5.2. Index Construction Based on HAC. In this section, we propose an index structure which is based on the hierarchical agglomerative clustering method. We name this index as HAC-tree. The construction algorithm of HAC-tree is also given.

Definition 3 (extract maximum vector). One assumes that $\mathrm{VS} = \{V_1, V_2, \ldots, V_m\}$ is a set of n-dimensional vectors. The extract maximum vector of VS is denoted as $V_{\max} = \overrightarrow{\max}\{V_1, V_2, \ldots, V_m\}$ and

$$V_{\max}[j] = \max\{V_1[j], V_2[j], \ldots, V_m[j]\}, \quad j \in \{1, 2, \ldots, n\}. \tag{8}$$

Definition 4 (HAC-tree node structure). Node u of HAC-tree is a five-element tuple $\langle \mathrm{VM}, \mathrm{PL}, \mathrm{PR}, \mathrm{FD}, \mathrm{sig} \rangle$, where u.VM is the *pruning vector*, u.PL and u.PR, respectively, point to the left and right child nodes of u, u.FD stores the unique identifier of a document, and u.sig stores a digest of the u.FD document.

In addition, u.VC is the *cluster center vector* of cluster C_u whose items are the documents represented by the leaf nodes of the subtree with node u as its root, and $u.N$ is the number of documents of C_u. It is noticeable that u.VC and $u.N$ are only used for HAC-tree construction but not needed to store in HAC-tree.

According to the types of node u, we give the detailed description of the HAC-tree node as follows:

(1) If u is a leaf node, u.PL = u.PR = ∅, u.FD stores an identifier of a document, u.VM and u.VC both store the vector of u.FD document, $u.N = 1$, and u.sig stores a digest of u.FD document which is used for search results verification and illustrated in Section 5.4.

(2) If u is an internal node, u.FD = ∅, u.sig = ∅, u.PL and u.PR, respectively, represent the left and right child nodes of u, $u.N = u.\mathrm{PL}.N + u.\mathrm{PR}.N$, and u.VM is the extract maximum vector of C_u, while u.VC is the cluster center of C_u. We have

$$u.\mathrm{VM} = \overrightarrow{\max}\{u.\mathrm{PL}.\mathrm{VM}, u.\mathrm{PR}.\mathrm{VM}\}, \tag{9}$$

$$u.\mathrm{VC} = \frac{u.\mathrm{PL}.N \times u.\mathrm{PL}.\mathrm{VC} + u.\mathrm{PR}.N \times u.\mathrm{PR}.\mathrm{VC}}{u.\mathrm{PL}.N + u.\mathrm{PR}.N}. \tag{10}$$

Definition 5 (pruning vector). For node u in HAC-tree, we name u.VM as the pruning vector of u which will be used to filter the noncandidate subtree and hence improve the efficiency of the multikeyword text search.

According to Definition 4, each node u in HAC-tree has two kinds of vectors. One is the *pruning vector* u.VM, which is stored in HAC-tree. It can be utilized to filter the noncandidate subtree for search efficiency improving. The other is the *cluster center vector* u.VC, which is not actually stored in HAC-tree but only used for HAC-tree construction.

Based on the definition of HAC-tree node structure and the hierarchical agglomerative clustering, we give the HAC-tree construction algorithm as in Algorithm 1.

In Algorithm 1, $H(k_f \parallel F_i)$ is a digest generating function for document which will be introduced in Section 5.4. We give an example of HAC-tree construction as shown in Figure 4. Here, we take two leaf nodes $u_{4,1}$, $u_{4,2}$ and one internal node $u_{3,1}$ to illustrate how the *BuildHACTree* algorithm works. The nodes of $u_{4,1}$ are leaf nodes; according to *BuildHACTree* algorithm, we know that $u_{4,1}$.PL = $u_{4,1}$.PR = $\emptyset$, $u_{4,1}$.FD stores the identifier of F_4, $u_{4,1}$.sig = $H(k_f \parallel F_4)$, and $u_{4,1}$.VM stores the vector of F_4 which is $(0.5, 0.8, 0.2, 0)$. Similarly, $u_{4,2}$.PL = $u_{4,2}$.PR = $\emptyset$, $u_{4,2}$.FD stores the identifier of F_8, $u_{4,2}$.sig = $H(k_f \parallel F_8)$, and $u_{4,2}$.VM stores the vector of F_8 which is $(0.6, 0.9, 0.1, 0)$. Then, we construct the parent node $u_{3,1}$ based on $u_{4,1}$ and $u_{4,2}$. Node $u_{3,1}$ is an internal node, so $u_{3,1}$.FD = $\emptyset$ and $u_{3,1}$.sig = $\emptyset$. Pointer $u_{3,1}$.PL points to the left child node $u_{4,1}$ and $u_{3,1}$.PR points to the right child node $u_{4,2}$. According to (9), we have $u_{3,1}$.VM = $(0.6, 0.9, 0.2, 0)$. Now, we complete the construction of $u_{3,1}$. Just following the above procedures, we can construct the HAC-tree from the bottom leaf nodes to the top root node, which is shown in Figure 4. For simplicity, the vectors of documents are not normalized.

According to the construction process of HAC-tree, we have the following lemmas.

Lemma 6. *Assume that $u \in I$ is an internal node where $u.VM[i] \neq 0$ where $i \in \{1, 2, \ldots, n\}$ and the subtree with the root u is denoted as I_u. Then one has the following:*

(1) *There exists at least one leaf node $v \in I_u$ satisfying that the keyword w_i appears in the corresponding document of v.*

(2) $\forall v \in I_u$ $(u.VM[i] \geqslant v.VM[i])$.

Proof. (1) We give the proof of contradiction for Lemma 6(1). We assume that there is no leaf node $v \in I_u$ satisfying that w_i appears in its corresponding document. It means that all the corresponding documents of leaf nodes in I_u do not have the keyword w_i. Then we have that for each leaf node $v \in I_u$, v.VM$[i] = 0$ holds. According to the pruning vector computation in the HAC-tree node structure definition, we can deduce u.VM$[i] = 0$ which is conflictive to the given condition u.VM$[i] \neq 0$. Hence, Lemma 6(1) is true.

(2) According to the pruning vector computation, for any $v \in I_u$, if l and r are the left and right child of v, respectively, we have v.VM$[i] = \max\{l$.VM$[i], r$.VM$[i]\}$ which means v.VM$[i] \geqslant l$.VM$[i]$ and v.VM$[i] \geqslant r$.VM$[i]$. Because u is the root of I_u, we have that $\forall v \in I_u$ $(u$.VM$[i] \geqslant v$.VM$[i])$. Therefore, Lemma 6(2) holds. □

Lemma 7. *For any leaf node $v \in I$, assume that u is an ancestor of v; the path from u to v is $u \to u_1 \to u_2 \to \cdots \to u_t \to v$ and Q is a query vector; then one has* $\text{score}(u.VM, Q) \geq \text{score}(u_1.VM, Q) \geq \cdots \geq \text{score}(u_t.VM, Q) \geq \text{score}(v.VM, Q)$.

Proof. Assume that u_p and u_q are two nodes of the path where u_p is the ancestor of u_q. According to Lemma 6, we can easily have u_p.VM$[i] \geq u_q$.VM$[i]$ where $i \in \{1, 2, \ldots, n\}$. Since elements in a pruning vector and a query vector are nonnegative, we have $\text{score}(u_p.\text{VM}, Q) \geq \text{score}(u_q.\text{VM}, Q)$. Thus, we can easily deduce $\text{score}(u.\text{VM}, Q) \geq \text{score}(u_1.\text{VM}, Q) \geq \cdots \geq \text{score}(u_t.\text{VM}, Q) \geq \text{score}(v.\text{VM}, Q)$. Lemma 7 holds. □

5.3. Search Algorithm

Lemma 8. *Assuming that $u \in I$ is an internal node, Q is a query vector and τ is a threshold. The subtree with the root u is denoted as I_u. If* $\text{score}(u.VM, Q) < \tau$, *then one has* $\forall v \in I_u$, $\text{score}(v.VM, Q) < \tau$.

Proof. For any node $v \in I_u$, there must be a leaf node $l \in I_u$ satisfying that v is in the path from u to l. According to Lemma 7, we have $\text{score}(v.\text{VM}, Q) \leq \text{score}(u.\text{VM}, Q)$. If $\text{score}(u.\text{VM}, Q) < \tau$ is given, then $\text{score}(v.\text{VM}, Q) < \tau$ must hold. Therefore, Lemma 8 is true. □

For a given query whose vector is Q, the goal of the multikeyword text search is to find top-k documents which are most relevant to the query. We assume that R is a list storing candidate top-k result documents and τ is the minimum relevance score between the documents of R and the query Q. For any node $u \in I$ which is the root of subtree I_u, if $\text{score}(u.\text{VM}, Q) < \tau$, then we have $\forall v \in I_u$, $\text{score}(v.\text{VM}, Q) < \tau$ according to Lemma 8. It means that all the leaf nodes of I_u must not be the candidate result documents. Obviously, it is unnecessary to search I_u when we perform the query Q on it. Hence, I_u can be directly pruned without further searching which improves the search efficiency. Based on this idea, we propose the noncandidate pruning depth-first search algorithm (NCP-DFS) as in Algorithm 2.

In Algorithm 2, $GetMinScoreNode(R, Q)$ is to obtain the minimum relevance score between the documents corresponding to the leaf nodes in R and the query Q. When Algorithm 2 is finished, the corresponding documents of leaf nodes in R are the final search result documents.

An example of NCP-DFS processing is also shown in Figure 4. A query with the vector $Q = \{0.8, 0.6, 0, 0\}$ is given. The query is to obtain the top 2 relevant documents with Q. The access path is shown by the dotted arrow. The leaf nodes $u_{4,1}$ and $u_{4,2}$ are firstly add to R; then the threshold $\tau = 0.88$ is calculated. Then, because $\text{score}(u_{3,2}.\text{VM}, Q)$ and $\text{score}(u_{2,2}.\text{VM}, Q)$ are both less than τ, the subtrees with the roots $u_{3,2}$ and $u_{2,2}$ are pruned. At the same time, NCP-DFS execution ends. The search result is $\{F_4, F_8\}$ whose corresponding leaf nodes in R are $u_{4,1}$ and $u_{4,2}$.

Input: the document set FS = $\{F_1, F_2, \ldots, F_m\}$ with the respective file description $\{1, 2, \ldots, m\}$ and document vector $\{V_1, V_2, \ldots, V_m\}$
Output: the HAC-tree I
Variables:
CPNS—a set variable for storing the currently processing tree nodes
NGNS—a set variable for storing the newly generated tree nodes
Procedures:
(1) **FOR EACH** $F_i \in$ FS **DO**
(2) Construct a new leaf node u for documents F_i where u.PL = u.PR = $\emptyset$, u.FD = i, u.VM = u.VC = V_i, $u.N = 1$ and u.sig = $H(k_f \parallel F_i)$;
(3) Add u into CPNS;
(4) **END FOR**
(5) **WHILE** |CPNS| > 1 **DO**
(6) **WHILE** |CPNS| > 1 **DO**
(7) (u_i, u_j) = FindMaxCR(CPNS); //find a pair of nodes which have the maximum relevance score
(8) Construct a new node u as the parent of u_i and u_j where u.PL = u_i, u.PR = u_j, u.FD = $\emptyset$, u.sig = $\emptyset$, $u.N = u_i.N + u_j.N$, u.VM = $\overrightarrow{\max}\{u_i.\text{VM}, u_j.\text{VM}\}$, $u.\text{VC} = (u_i.N \times u_i.\text{VC} + u_j.N \times u_j.\text{VC})/(u_i.N + u_j.N)$;
(9) Add u into NGNS and delete u_i and u_j from CPNS;
(10) **END WHILE**
(11) **IF** |NGNS| > 0 **THEN**
(12) Merge NGNS into CPNS and then clear NGNS;
(13) **END IF**
(14) **END WHILE**
(15) **RETURN** the only node left in CPNS which is the root of the HAC-tree I;

Algorithm 1: *BuildHACTree*(FS).

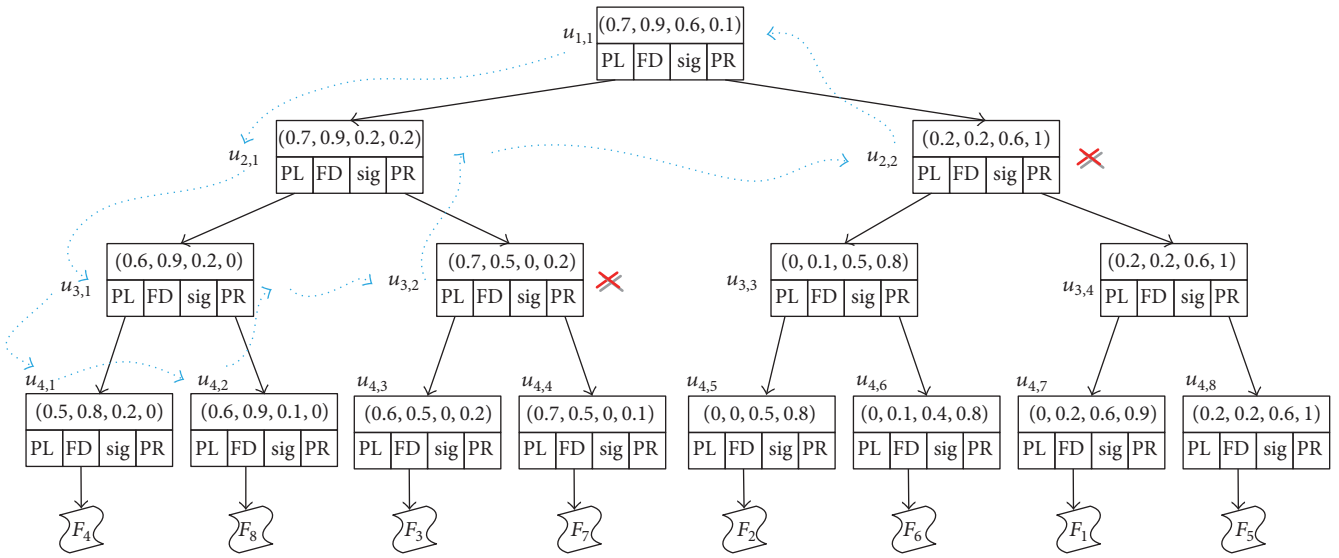


Figure 4: An example of HAC-tree where FS = $\{F_1, F_2, \ldots, F_8\}$ and the cardinality of the keyword dictionary is $n = 4$.

According to the relevance score definition in Section 3, the time complexity of a relevance score computation between a document and a query is $O(n)$. Lemma 2 shows that the height of the HAC-tree is $\lceil \log_2 m \rceil + 1$. The number of leaf nodes is denoted as ω, whose corresponding documents contain one or more interested keywords in the query. Then we have that the time complexity of NCP-DFS is $O(\omega \cdot n \cdot \log m)$. It is noticeable that the real search time is less than $\omega \cdot n \cdot \log m$ distinctly. The reasons are as follows: (1) due to subtree pruning in advance, many leaf nodes will not be accessed even though one or more keywords are contained. (2) The accessing paths of some different leaf nodes in the same subtree share the mutual traversed parts. (3) The documents with higher relevance scores are clustered

Input: r is the root of the current searched sub-tree, Q is the vector of a query and k is the number of requested documents.
Output: the top-k documents
Variables:
u, u'—a local variable for tree node;
τ—a global variable for storing a relevance score threshold which is initialized that $\tau = 0$;
R—a list variable for storing the leaf node containing candidate result documents which are initialized that $R = \emptyset$.
When NCP-DFS is finished, R contains the leaf nodes which stores the top-k result documents.
Procedures:
(1) $u = r$;
(2) **IF** u is a leaf node **THEN**
(3) **IF** $|R| < k - 1$ **THEN**
(4) Add u into R;
(5) **ELSE IF** $|R| == k - 1$ **THEN**
(6) Add u into R;
(7) $\tau = GetMinScoreNode(R, Q)$; //obtain the kth relevance score in R as a threshold
(8) **ELSE**
(9) **IF** score$(u.\mathrm{VM}, Q) > \tau$ **THEN**
(10) $u' = GetMinScoreNode(R, Q)$;
(11) Delete u' from R and add u into R;
(12) $u' = GetMinScoreNode(R, Q)$;
(13) $\tau =$ score$(u'.\mathrm{VM}, Q)$;
(14) **END IF**
(15) **END IF**
(16) **ELSE** //if u is the internal node
(17) **IF** score$(u.\mathrm{VM}, Q) > \tau$ **THEN**
(18) NCP-DFS$(u.\mathrm{PL}, Q, k)$;
(19) NCP-DFS$(u.\mathrm{PL}, Q, k)$;
(20) **END IF**
(21) **END IF**

Algorithm 2: NCP-DFS(r, Q, k).

together and more subtrees will be filtered which reduces the accessed leaf nodes furtherly. In addition, we can easily have that NCP-DFS is able to find the accurate search results.

5.4. Search Result Verification. The network instability or potential hardware/software errors may result in tampering or even loss of documents, which could make the search results incomplete and/or incorrect. Therefore, in each leaf node of HAC-tree, we embed a digest for the corresponding document. Once a query is performed in CS, digests of the result documents are used to construct a verification object (VO) for DU. When DU receives the returned search result data from CS, DU obtains the result documents and verifies their completeness and correctness by using the verification object.

Definition 9 (document digest). One assumes that u is a leaf node of HAC-tree, whose corresponding document is F_u. The attribute u.sig is a document digest of F_u and

$$u.\mathrm{sig} = H\left(k_f \parallel F_u\right), \tag{11}$$

where $H(\cdot)$ is a Hash Message Authentication Coding (HMAC), for example, HMAC-SHA1, k_f is a key only shared by DO and DU, and $\parallel$ denotes concatenation.

Definition 10 (verification object). One assumes that the set of digests of the documents in a search result is denoted as G. The verification object (VO) of the search results is constructed as follows:

$$\mathrm{VO} = \bigoplus_{g_i \in G} g_i. \tag{12}$$

The completeness and correctness of the search results are verified by Algorithm 3.

Algorithm 3 indicates that, under the honest-but-curious model, if a search result is useless because one or more documents of the search result are dropped or falsified for some reasons, DU will detect it through the search result verifications.

The aforementioned works discuss the HAC-tree construction and the corresponding multikeyword text search algorithm NCP-DFS. However, these are based on plaintext. In the next section, we will use secure inner product operation to encrypt HAC-tree and give two kinds of multikeyword text search schemes over encrypted cloud data.

6. Secure Multikeyword Text Search Schemes

In this section, two secure search schemes, based on the framework of MUSE (as shown in Section 4.5), are given to resist the known ciphertext threat model and known background threat model, respectively. One is the basic multikeyword text search over encrypted (BMUSE) cloud data and

Input: R is a set of the result documents, VO is the verification object from CS and k is the number requested documents
Output: Whether R is correct and complete
Variables: G is a set variable for storing document digests
Procedures:
(1) **IF** $|R| \neq k$ **THEN**
(2) **RETURN** FALSE; //R is incomplete
(3) **ELSE**
(4) **FOR EACH** $F_i \in R$ **DO**
(5) Add $H(k_f \parallel F_i)$ into G;
(6) **END FOR**
(7) **IF** VO $\neq \bigoplus_{g_i \in G} g_i$ **THEN**
(8) **RETURN** FALSE; //R is incorrect
(9) **END IF**
(10) **END IF**
(11) **RETURN** TRUE; //R is complete and correct

Algorithm 3: $VerifyResult(R, \text{VO}, k)$.

the other is enhanced secure multikeyword text search over encrypted (EMUSE) cloud data.

6.1. BMUSE Scheme. We use the *BuildHACTree* algorithm, *NCP-DFS* algorithm, *VerifyResult* algorithm, and secure inner product operation to construct the BMUSE scheme whose details are shown as follows.

$GenKey(1^{l(n)})$. In this initialization phase, DO generates the secure key SK, which is a four-element tuple as (S, M_1, M_2, k_f) where S is an n-bit random vector; M_1 and M_2 are two random $n \times n$ invertible matrices; k_f is a key for symmetric encryption. SK is only shared between DO and DU while CS has no idea of it.

$BuildIndex(FS, SK)$. According to *BuildHACTree* algorithm, for each vector u.VM stored in the node u, the splitting procedure is applied, and it is split into two random vectors as $\{V_u', V_u''\}$ with S as a splitting indicator. The split rule is shown as

$$\begin{aligned} V_u'[i] + V_u''[i] &= u.\text{VM}[i] \quad S[i] = 0 \\ V_u'[i] = V_u''[i] &= u.\text{VM}[i] \quad S[i] = 1. \end{aligned} \tag{13}$$

The encrypted vectors of u.VM are generated as $\{M_1^T V_u', M_2^T V_u''\}$. For each node in HAC-tree, u.VM is replaced with $\{M_1^T V_u', M_2^T V_u''\}$; then encrypted tree index $\tilde{I}$ is generated.

$GenTrapdoor(S_q, k, SK)$. Assume that $S_q = \{w_1, w_2, \ldots, w_t\}$ is a set of interested keywords. The corresponding vector of S_q is generated as Q, each dimension of which is computed as (14). And then Q is normalized.

$$Q[i] = \begin{cases} \text{IDF}_i, & w_i \in S_q \\ 0, & \text{other.} \end{cases} \tag{14}$$

Subsequently, Q is split into two random vectors as $\{Q', Q''\}$ with the splitting indicator S, and the split rule is shown as (15). Then the encrypted vectors of Q are generated as $\{M_1^{-1}Q', M_2^{-1}Q''\}$. After that, DU transmits the trapdoor T_Q to CS, which consists of $\{M_1^{-1}Q', M_2^{-1}Q''\}$ and k.

$$\begin{aligned} Q'[i] = Q''[i] &= Q[i], \quad S[i] = 0 \\ Q'[i] + Q''[i] &= Q[i], \quad S[i] = 1. \end{aligned} \tag{15}$$

$DoSearch(\tilde{I}, T_Q, k)$. CS executes NCP-DFS algorithm to obtain the search result R and construct a verification object VO. Then CS returns R together with VO to DU. The relevance score between the encrypted vector of u.VM and the encrypted vector of Q is calculated by (16). Note that the relevance score calculated from encrypted vectors is equal to that of unencrypted vectors as follows:

$$\begin{aligned} &\text{score}\left(\left\{M_1^T V_u', M_2^T V_u''\right\}, \left\{M_1^{-1}Q', M_2^{-1}Q''\right\}\right) \\ &= \left(M_1^T V_u'\right)^T \cdot M_1^{-1}Q' + \left(M_2^T V_u''\right)^T \cdot M_2^{-1}Q'' \\ &= V_u' \cdot Q' + V_u'' \cdot Q'' = u.\text{VM} \cdot Q. \end{aligned} \tag{16}$$

$Verify(R, VO, SK)$. DU utilizes the key k_f to decrypt the search result and verifies it through the *VerifyResult* algorithm.

6.2. EMUSE Scheme. According to [3, 7], BMUSE may leak the privacy of keywords. The cloud server is able to link the same search requests according to the same visited path and the same relevance scores. So CS may find some high-frequency keywords and deduce whether a certain keyword exists in a document by analyzing the correlation of trapdoors. In consideration of this, it is a practical and effective method to extend dimension by adding some phantom terms into document vectors and query vectors to break such equality, and then the keyword privacy and trapdoor unlinkability will be protected. Following the above idea, we propose the enhanced MUSE which is named EMUSE.

$GenKey(1^{l(n)})$. CS generates the secure key SK, which is four-element tuple, $SK = (S, M_1, M_2, k_f)$. S is $(n + e)$-bit random vector; M_1 and M_2 are two random $(n+e) \times (n+e)$ invertible matrices; k_f is a key for symmetric encryption.

$BuildIndex(FS, SK)$. Through adding the phantom terms, we extend u.VM from n dimension to $(n + e)$ dimension. Each element of phantom terms is filled by a random value ϑ_i, $\vartheta_i \sim U(\mu' - \theta, \mu' + \theta)$. Then we adopt the same vector encryption method as BMUSE to encrypt u.VM for every vector in the HAC-tree.

$GenTrapdoor(S_q, k, SK)$. We extend Q from n dimension to $(n + e)$ dimension to generate the query vector. A part of elements in the extended dimensions of Q are set to 1 and the left are set to 0.

$DoSearch(\widetilde{I}, T_Q, k)$ and $Verify(R, \text{VO}, \text{SK})$ are the same as the ones in Section 6.1.

According to the detailed steps of the above schemes, we discuss the deployment of our schemes. Obviously, DO is the manager for deploying the schemes. DO generated security information SK firstly which is utilized to encrypt the documents and generate the corresponding secure index. After that the encrypted documents and index are outsourced in CS. To process secure multikeyword search, the search algorithm on the basis of the outsourced data should also be deployed in CS by DO. Meanwhile, the trapdoor generating algorithm designed by DO is granted to the authorized DU together with SK. When the authorized DU performs a search, a trapdoor is firstly generated and submitted to CS. After that, CS performs the secure multikeyword search algorithm designed by DO and returns the corresponding encrypted documents and verification object to DU. And DU applies the decryption to get the plaintext search result.

7. Security Analysis

7.1. Privacy-Preserving Analysis. In this section, we analyze the privacy-preserving property of BMUSE and EMUSE, which depends on the security of secure inner production firstly proposed in the secure k-nearest neighbor (kNN) technique [13]. All vectors in HAC-tree are encrypted by random invertible matrices and all documents in FS are encrypted by symmetric encryption. According to [3], CS is impossible to deduce the concrete random matrices on the basis of grasping existing data, including the encrypted documents, the encrypted HAC-tree, and the trapdoor of queries. That is to say, the keys for HAC-tree encryption are secure. Besides, the other security information in SK, such as the symmetric encryption key and splitting indicator, is only shared between DO and DU but private to CS. Thus, BMUSE can protect the confidentiality of documents, index, and queries. Therefore, BMUSE can resist the attack of known ciphertext threats. However, if the given interested multikeyword are the same, the corresponding generated trapdoors and the access paths are the same too. BMUSE cannot resist the attack of known background threats, which means that BMUSE may leak the trapdoor unlinkability [7] and keyword privacy.

In EMUSE, we add phantom terms on the basis of BMUSE, which are equivalent to adding random values to the relevance scores between documents and queries. The security is enhanced. Therefore, EMUSE is able to withstand known ciphertext threats. That is to say, EMUSE satisfies all the privacy-preserving requirements described in Section 4.4.

We denote that the document vector V_d is expanded to V_e through adding phantom terms, and the query vector Q is expanded to Q_e. We have $V_e \cdot Q_e = V_d \cdot Q + \sum_{i=1}^{0.5e} \vartheta_i$, so the added random value $\xi = \sum_{i=1}^{0.5e} \vartheta_i$. According to the central-limit theorem, we have that if $\vartheta_i \sim U(\mu' - \theta, \mu' + \theta)$, then $\xi \sim N(\mu, \sigma^2)$, where $\mu = 0.5e\mu'$, $\sigma = \sqrt{(1/6)e} \cdot \theta$.

Because of adding a random value ξ to the relevance score between a document and a query, the accuracy of the search result slightly decreases but the privacy preservation is enhanced. Therefore, we need to control ξ to get the balance of the accuracy and security. Practically, we set $\mu = 0$ and balance the accuracy and security by adjusting the standard deviation σ.

7.2. Completeness Analysis. In the HAC-tree leaf node, u.sig is document digest of document F_i represented by u.FD constructed by the key of k_f and the context of F_i based on HMAC. Both the avalanche effect [37] and one-way irreversibility guarantee the cryptographic strength of the HMAC, thus ensuring the safety of u.sig.

Assume that $r = \{f_1, f_2, \ldots, f_k\}$ and vo are returned from CS where r is the result document set and vo is the verification object. After receiving the search result, DU recalculates the document digests as $\text{sig}_1, \text{sig}_2, \ldots, \text{sig}_k$ and reconstructs the verification object, denoted as vo$'$. If the content of f_i is tampered into f_i', then vo$' \neq$ vo will happen. Besides, for each document digest, it is constructed by the key of k_f and the plaintext of F_u. Both of them are private to CS. Hence, it is impossible for CS to falsify a correct document digest successfully without being detected. And MUSE is capable of supporting the search result completeness verification.

8. Performance Evaluation

In this section, we evaluate the performance of our proposed scheme MUSE and compare it with the schemes presented in [7, 8] which are denoted as DMRS and MRSE-HCI, respectively. We implement MUSE, DMRS, and MRSE-HCI and perform evaluations on the precision (Δ_p) and search time (Δ_t) on the real data set of NSF Research Award Abstracts provided by UCI [38]. We use BMUSE, BDMRS, and BMRSE-HCI to represent the basic schemes of MUSE, DMRS, and BMRSE-HCI, respectively. Meanwhile, we use EMUSE, EDMRS, and EMRSE-HCI to represent the enhanced schemes of MUSE, DMRS, and BMRSE-HCI, respectively.

The real dataset includes about 129000 abstracts, from which we random choose 20000 abstracts as our experimental data and extract about 10000 keywords. The experimental hardware environment is Intel Xeon E3-1225v, 3.3 GHz 4 core CPU, 16 G memory, and 1 T hard disk; and

Table 1: Default evaluation parameters.

Para.	μ	σ	m	n	k	t	TH
Val.	0	0.02	20000	10000	20	10	100

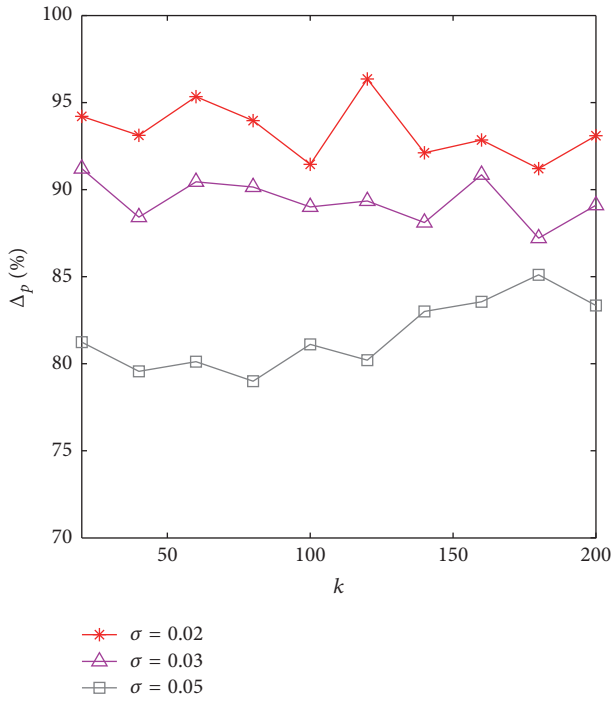


Figure 5: The impact of k on precision by choosing different standard deviation σ.

software environment is CentOS6.4 server operating system and Eclipse development platform. Default parameters are summarized in Table 1. Parameter t denotes the number of keywords of interest. TH donates the maximum number of cluster size in MRSE-HCI.

8.1. Search Precision Evaluation. Due to adding phantom terms in EMUSE, the relevance score between a document and a query is affected by the random value ξ, $\xi \sim N(0, \sigma^2)$. In order to evaluate the impact of σ on the accuracy of the search results, we adopt the definition of precision in [3] which is defined as $\Delta_p = k'/k$, where k' is the number of the real top-k result documents that are returned by CS.

Figure 5 shows that the precisions fluctuate within a small scope when $\sigma = 0.02$, $\sigma = 0.03$, and $\sigma = 0.05$, respectively. Note that the smaller the standard deviation σ is, the higher the precision becomes. In the experiment, when σ is 0.02, 0.03, and 0.05, the value of Δ_p is 93.37%, 89.39%, and 81.62%, respectively. The reason is that when σ grows, the impact of ξ on relevance score between a document and a query becomes greater. In our proposed scheme, DU can balance the precision and security by adjusting the standard deviation σ. The default value of σ is 0.02 in the experiment.

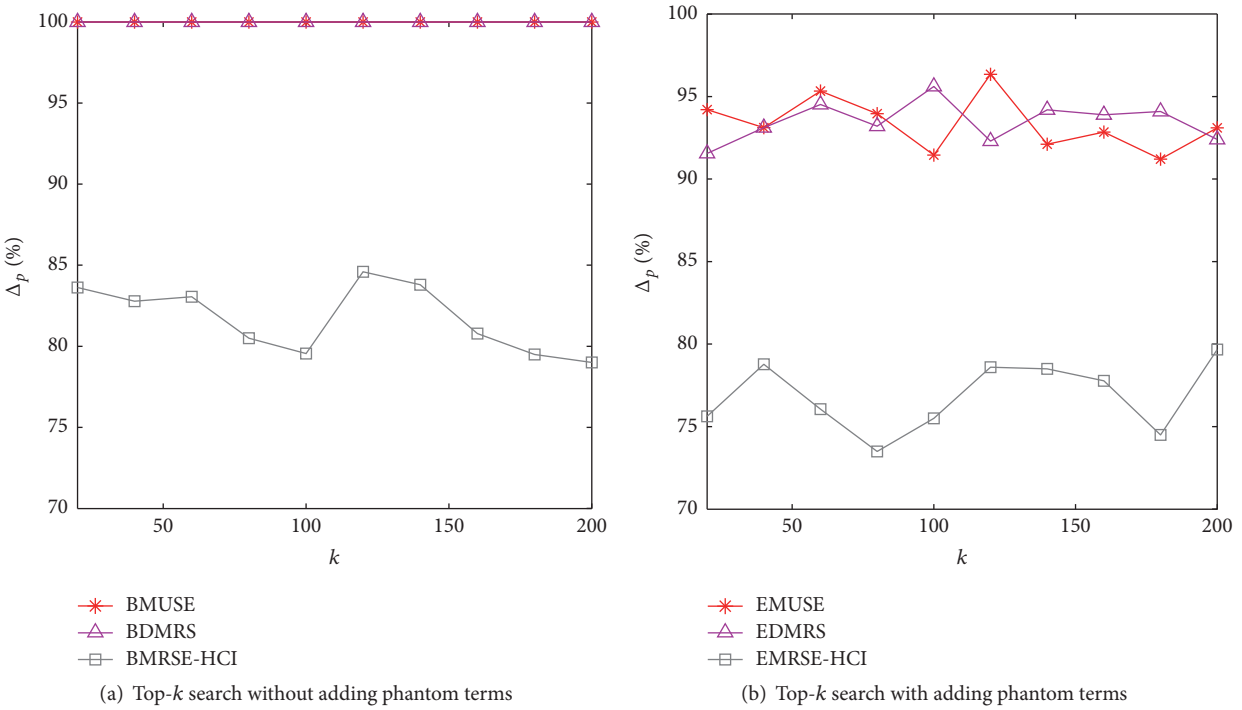
As shown in Figures 6(a) and 6(b), for MUSE, DMRS, and MRSE-HCI, Δ_p does not change significantly, when k increases from 20 to 200. However, no matter with or without adding phantom terms, the MUSE's Δ_p is significantly larger than MRSE-HCI and approximately equal to DMRS, which means that MUSE outperforms MRSE-HCI in terms of accuracy. Δ_p of BMUSE is 100% which is the same as BDMRS because both BMUSE and BDMRS perform the exact calculation of document vectors and query vectors. The precision of BMRSE-HCI is about 81.71% which is obviously lower than the others. The reason is that BMRSE-HCI clusters the relevant documents into the same cluster by dynamic k-means during index construction, and then the search involves only the most relevant subclusters, which will cause some more relevant documents to be ignored, and inaccurate search results are generated. Due to adding phantom terms, the EMUSE's Δ_p is no longer 100% and about 93.37%. Compared with EMRSE-HCI, the EMUSE's Δ_p is still 16.52% higher. The reason is the same as the basic scheme. In conclusion, the experimental result demonstrates that MUSE can achieve higher accuracy than MRSE-HCI and the similar accuracy with DMRS both in the basic and in enhanced scheme.

8.2. Search Time Evaluation. For the convenience of description, we use MRSE-HCI-100 and MRSE-HCI-300 to represent the maximum numbers of cluster size TH = 100 and TH = 300 in MRSE-HCI, respectively. In this section, we evaluate and analysis the impact of m, k, t on Δ_t in the basic and enhanced scheme.

(1) The Impact of m. As shown in Figures 7(a) and 7(b), the search time Δ_t of MUSE is obviously lower than DMRS in both basic and enhanced scheme. Figures 7(a) and 7(b) both reveal that Δ_t in MUSE and DMRS increase with m growing up. In addition, Δ_t of MUSE is about half of DMRS for the same dataset. The reason is that the documents with high relevance are clustered in HAC-tree. HAC-tree is helpful to maintain and utilize the category relationship to perform efficient and accurate text search on encrypted documents. This feature can ensure the efficiency of NCP-DFS algorithm. However, DMRS does not use the relationship between documents, so it is obviously slower than MUSE.

Besides, as shown in Figures 7(a) and 7(b), Δ_t of MRSE-HCI grows very slowly as m grows. For the basic and enhanced schemes of MRSE-HCI, the value of TH has great influence on search efficiency. When TH is 100, Δ_t of MRSE-HCI is lower than others. And when TH is 300, Δ_t of MUSE is firstly lower than MRSE-HCI but the former becomes larger than the later as m grows. The improvement of efficiency in MRSE-HCI is at the expense of sacrificing the search result precision as shown in Figures 6(a) and 6(b).

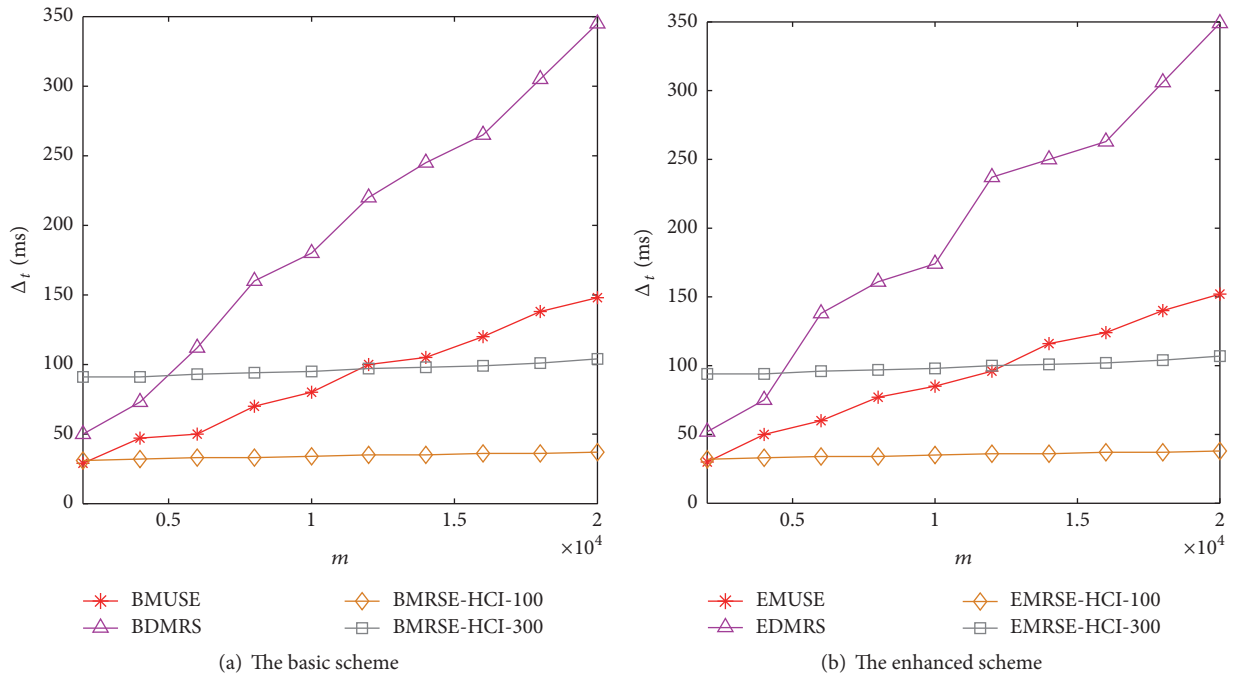
In addition, we have that the plot of EMUSE is very likely to the plot of BMUSE, and the search time of EMUSE is slightly higher than BMUSE, about 5.67%. It indicates that



(a) Top-k search without adding phantom terms

(b) Top-k search with adding phantom terms

FIGURE 6: The impact of k on the precision of search results.



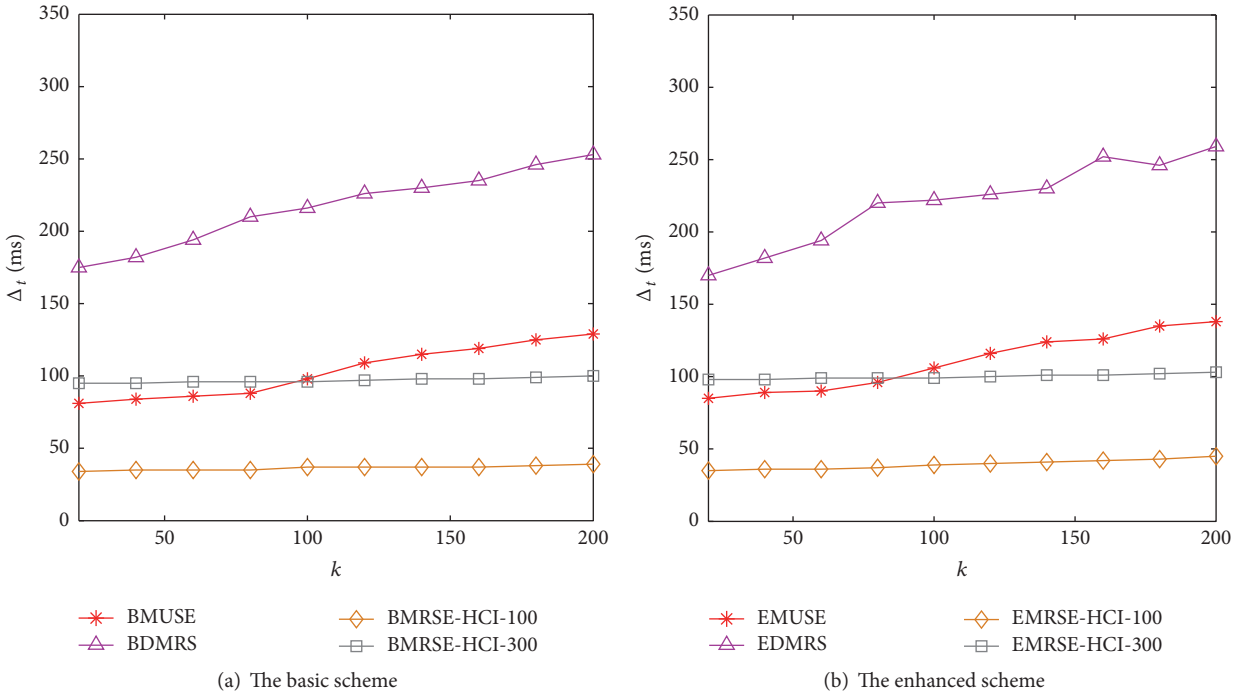
(a) The basic scheme

(b) The enhanced scheme

FIGURE 7: The impact of m on search efficiency.

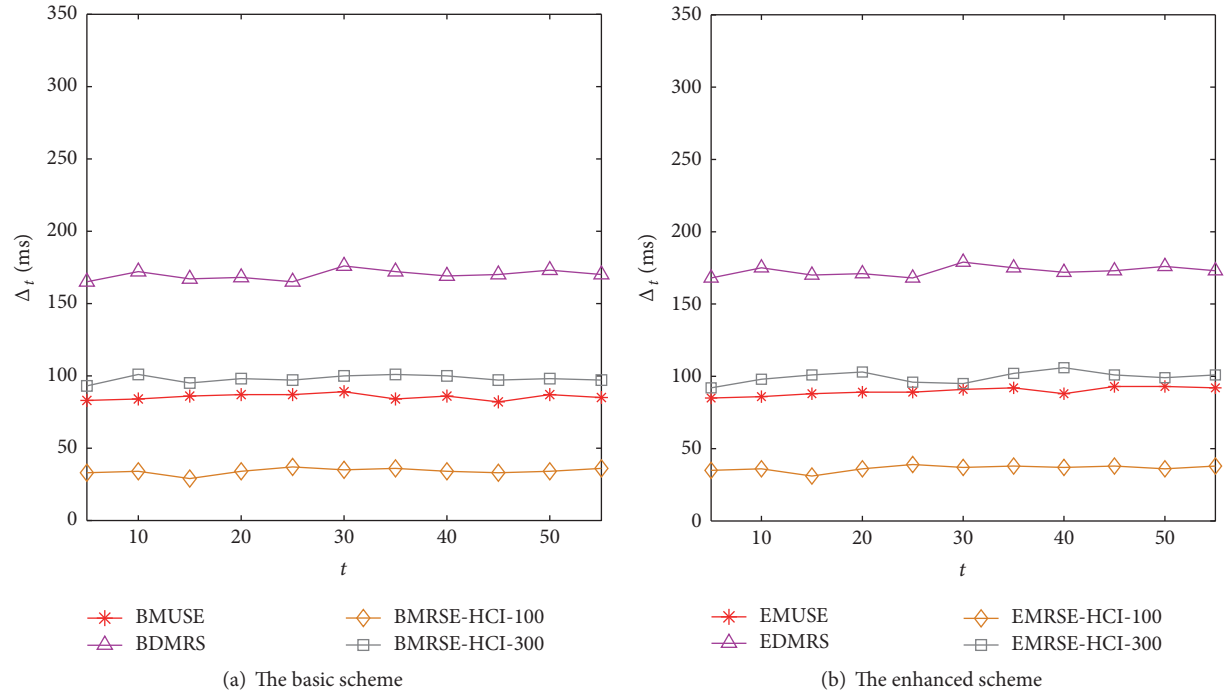
there is just a little impact on the search time by adding phantom terms into vectors.

(2) The Impact of k. As shown in Figures 8(a) and 8(b), no matter before or after adding phantom terms, Δ_t of all schemes increase as k grows. The reason is that more result documents will be determined as k grows, which consumes more time. The increments of Δ_t in MUSE and DMRS are more obvious than MRSE-HCI. The search times of BMUSE and EMUSE are about 47.71% and 40.30% less than BDMRS and EDMRS, respectively. While Δ_t of MRSE-HCI is less than the other schemes. But the better performance



(a) The basic scheme (b) The enhanced scheme

Figure 8: The impact of k on search efficiency.



(a) The basic scheme (b) The enhanced scheme

Figure 9: The impact of t on search efficiency.

in search efficiency of MRSE-HCI sacrifices the search result accuracy, and the reduction of accuracy may not satisfy users' expectations.

(3) The Impact of t. According to Figure 9, Δ_t of MUSE, DMRS, and MRSE-HCI have little changes no matter with or without adding phantom terms. This means that the search times of MUSE, DMRS, and MRSE-HCI are affected slightly by the number of keywords of interest. As shown in Figure 9(a), in terms of efficiency, BMUSE is better than BDMRS and saves about 51.12% in Δ_t. It also reveals that BMUSE is superior to BMRSE-HCI-300 and inferior to

BMRSE-HCI-100. BMUSE puts accuracy as a prerequisite to obtaining efficiency as much as possible. In contrast, BMRSE-HCI-100 improves the efficiency of result retrieval by the cost of reducing accuracy. In Figure 9(b), Δ_t of EMUSE is almost a half of EDMRS but during between EMRSE-HCI-100 and EMRSE-HCI-300. The reason is the same as the basic scheme.

According to (1), (2), and (3), the experimental results demonstrate that MUSE can achieve higher search efficiency than DMRS. While MUSE may be less efficient than MRSE-HCI depending on the setting of the maximum number of cluster size parameter in MRSE-HCI.

Compared to DMRS and MRSE-HCI on a real dataset, the experimental results show that (i) MUSE outperforms DMRS in terms of efficiency and achieves the similar accuracy with DMRS; and (ii) MUSE consistently gives more accurate results than MRSE-HCI while it may be less efficient than MRSE-HCI, depending on the choices of parameter values in MRSE-HCI.

9. Conclusion

The multikeyword text search over encrypted data is a common problem in cloud computing services. There are urgent requirements in such important fields as e-mail, electronic medical records, and bank transaction information, which is a hot issue in cloud computing service. It is challenge to ensure the efficiency of the search under the premise of ensuring the accuracy of the search results. In this paper, we propose an efficient and accurate verifiable privacy-preserving multikeyword text search over encrypted cloud data based on hierarchical agglomerative clustering called MUSE. According to hierarchical agglomerative clustering, we construct a binary tree structure as the index, which is named HAC-tree. Based on HAC-tree, we provide a non-candidate pruning depth-first search algorithm to improve search efficiency. Besides, we utilize secure inner product algorithm against two threat models. The experimental results show that the scheme proposed in this paper has better comprehensive performance in terms of efficiency and accuracy compared with the existing methods.

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 Grants nos. 61402014, 61572263, 61672297, and 61472193; the Natural Science Foundation of Jiangsu Province under Grant nos. BK20151511, BK20141429, and BK20161516; the Project of Natural Science Research of Jiangsu University under Grant 15KJB520027; the Postdoctoral Science Foundation of China under Grants nos. 2015M581794 and 2016M601859; the Postdoctoral Science Foundation of Jiangsu Province under Grant no. 1501023C; NUPTSF (NY214127); the Six Talent Peaks Project in Jiangsu Province (no. JNHB-062); the Natural Science Foundation of Anhui Province under Grant no. 1608085MF127; the Chuzhou University Scientific Research Fund under Grant no. 2015QD02.

References

[1] S. Grzonkowski, P. M. Corcoran, and T. Coughlin, "Security analysis of authentication protocols for next-generation mobile and CE cloud services," in *Proceedings of the IEEE International Conference on Consumer Electronics*, pp. 83–87, Berlin, Germany, 2011.

[2] S. Kamara and K. Lauter, "Cryptographic cloud storage," in *Proceedings of the International Conference on Financial Cryptography and Data Security*, pp. 136–149.

[3] 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 IEEE INFOCOM*, pp. 829–837, Shanghai, China, April 2011.

[4] 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 the 8th ACM SIGSAC symposium*, pp. 71–82, Hangzhou, China, 2013.

[5] R. Li, Z. Xu, W. Kang, K. C. Yow, and C.-Z. Xu, "Efficient multi-keyword ranked query over encrypted data in cloud computing," *Future Generation Computer Systems*, vol. 30, no. 1, pp. 179–190, 2014.

[6] C. Orencik, A. Selcuk, E. Savas, and M. Kantarcioglu, "Multi-Keyword search over encrypted data with scoring and search pattern obfuscation," *International Journal of Information Security*, vol. 15, no. 3, pp. 251–269, 2016.

[7] Z. Xia, X. Wang, X. Sun, Q. Liu, and Q. Wang, "A secure and dynamic multi-keyword ranked search scheme over encrypted cloud data," *IEEE Transactions on Parallel and Distributed Systems*, 2015.

[8] 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.

[9] C. Gentry, *A fully homomorphic encryption scheme [Ph.D dissertation]*, Stanford University, Calif, USA, 2009.

[10] D. Boneh, G. Di Crescenzo, R. Ostrovsky, and G. Persiano, "Public key encryption with keyword search," in *International Conference on the Theory and Applications of Cryptographic Techniques*, pp. 506–522, Springer Berlin Heidelberg, Berlin, Germany, 2004.

[11] D. Cash, S. Jarecki, C. Jutla et al., "Highly-scalable searchable symmetric encryption with support for Boolean queries," in *Advances in Cryptology*, pp. 353–373, Springer Berlin Heidelberg, Berlin, Germany, 2013.

[12] S. Kamara, C. Papamanthou, and T. Roeder, "Dynamic searchable symmetric encryption," in *Proceedings of the 2012 ACM Conference on Computer and Communications Security, CCS 2012*, pp. 965–976, usa, October 2012.

[13] W. K. Wong, D. W. Cheung, B. Kao, and N. Mamoulis, "Secure kNN computation on encrypted databases," in *Proceedings of the International Conference on Management of Data and 28th Symposium on Principles of Database Systems (SIGMOD-PODS '09)*, pp. 139–152, Providence, RI, USA, July 2009.

[14] E.-J. Goh, "Secure indexes," Cryptology ePrint Archive 2003/216, 2003.

[15] H. Li, H. Ren, H. Yao, H. Chen, G. Xu, and Y. Dai, "Diverse multi-keyword ranked search over encrypted cloud data supporting range query," in *Proceedings of the IEEE/CIC International Conference on Communications in China, ICCC 2015*, China, November 2015.

[16] Z. Fu, X. Sun, Z. Xia, L. Zhou, and J. Shu, "Multi-keyword ranked search supporting synonym query over encrypted data in cloud computing," in *Proceedings of the 2013 IEEE 32nd International Performance Computing and Communications Conference, IPCCC 2013*, USA, December 2013.

[17] Z. Fu, X. Sun, N. Linge, and L. Zhou, "Achieving effective cloud search services: Multi-keyword ranked search over encrypted cloud data supporting synonym query," *IEEE Transactions on Consumer Electronics*, vol. 60, no. 1, pp. 164–172, 2014.

[18] Z. Fu, K. Ren, J. Shu, X. Sun, and F. Huang, "Enabling personalized search over encrypted outsourced data with efficiency improvement," *Parallel and Distributed Systems*, vol. 27, no. 9, pp. 2546–2559, 2015.

[19] M. Alewiwi, C. Orencik, and E. Savaş, "Efficient top-k similarity document search utilizing distributed file systems and cosine similarity," *Cluster Computing*, vol. 19, no. 1, pp. 109–126, 2016.

[20] M. Strizhov and I. Ray, "Multi-keyword similarity search over encrypted cloud data," *IACR Cryptology ePrint Archive*, vol. 137, 2015.

[21] H. Li, Y. Yang, T. H. Luan, X. Liang, L. Zhou, and X. S. Shen, "Enabling fine-grained multi-keyword search supporting classified sub-dictionaries over encrypted cloud data," *IEEE Transactions on Dependable and Secure Computing*, vol. 13, no. 3, pp. 312–325, 2016.

[22] C. Orencik, M. Kantarcioglu, and E. Savas, "A practical and secure multi-keyword search method over encrypted cloud data," in *Proceedings of the 2013 IEEE 6th International Conference on Cloud Computing, CLOUD 2013*, pp. 390–397, Santa Clara, Calif, USA, July 2013.

[23] D. Mashauri, R. Li, H. Han, X. Gu, Z. Xu, and C.-Z. Xu, "Adaptive multi-keyword ranked search over encrypted cloud data," *Lecture Notes of the Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering, LNICST*, vol. 163, pp. 3–13, 2016.

[24] M. Strizhov and I. Ray, "Secure multi-keyword similarity search over encrypted cloud data supporting efficient multi-user setup," *Transactions on Data Privacy*, vol. 9, no. 2, pp. 131–159, 2016.

[25] D. Thiyagaraja and R. Ganesan, "Cryptographically imposed model for efficient multiple keyword-based search over encrypted data in cloud by secure index using bloom filter and false random bit generator," *I. J. Network Security*, vol. 19, no. 3, pp. 413–420, 2017.

[26] D. X. Song, D. Wagner, and A. Perrig, "Practical techniques for searches on encrypted data," in *Proceedings of the IEEE Symposium on Security and Privacy (S&P '00)*, pp. 44–55, IEEE, Berkeley, Calif, USA, May 2000.

[27] Y.-C. Chang and M. Mitzenmacher, "Privacy preserving keyword searches on remote encrypted data," in *Proceedings of the Third International Conference on Applied Cryptography and Network Security, ACNS 2005*, pp. 442–455, usa, June 2005.

[28] D. Boneh, D. Di Crescenzo, R. Ostrovsky, and G. Persiano, "Public Key Encryption with Keyword Search," in *Proceedings of the Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, 2004.

[29] Z.-Y. Shao and B. Yang, "On security against the server in designated tester public key encryption with keyword search," *Information Processing Letters*, vol. 115, no. 12, pp. 957–961, 2015.

[30] 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.

[31] 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.

[32] Z. Chen, C. Wu, D. Wang et al., "Conjunctive keywords searchable encryption with efficient pairing, constant ciphertext and short trapdoor," *PAISI*, pp. 176–189, 2012.

[33] D. Cash, S. Jarecki, C. S. Jutla, H. Krawczyk, M.-C. Roşu, and M. Steiner, "Highly-scalable searchable symmetric encryption with support for Boolean queries," in *Advances in Cryptology – CRYPTO 2013*, R. Canetti and J. A. Garay, Eds., vol. 8042 of *Lecture Notes in Computer Science*, pp. 353–373, Springer, Berlin, Heidelberg, 2013.

[34] H. Pang, J. Shen, and R. Krishnan, "Privacy-preserving similarity-based text retrieval," *ACM Transactions on Internet Technology*, vol. 10, no. 1, article 4, 2010.

[35] 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.

[36] Z. Wan and R. H. Deng, "VPSearch: achieving verifiability for privacy-preserving multi-keyword search over encrypted cloud data," *IEEE Transactions on Dependable and Secure Computing*, vol. PP, no. 99, 1 page.

[37] V. R. Pallipamu, K. Thammi Reddy, and P. Suresh Varma, "ASH-160: A novel algorithm for secure hashing using geometric concepts," *Journal of Information Security and Applications*, vol. 21, pp. 52–63, 2015.

[38] M. Lichman, *UCI Machine Learning Repository*, University of California, School of Information and Computer Science, Irvine, Calif, USA, 2013, http://archive.ics.uci.edu/ml.