A Practical Framework for Secure Document Retrieval in Encrypted Cloud File Systems

Junsong Fu, Na Wang, Baojiang Cui, Bharat K. Bhargava, *Life Fellow, IEEE*

***Abstract*—With the development of cloud computing, more and more data owners are motivated to outsource their documents to the cloud and share them with the authorized data users securely and flexibly. To protect data privacy, the documents are generally encrypted before being outsourced to the cloud and hence their searchability decreases. Though many privacy-preserving document search schemes have been proposed, they cannot reach a proper balance among functionality, flexibility, security and efficiency. In this paper, a new encrypted document retrieval system is designed and a proxy server is integrated into the system to alleviate data owner's workload and improve the whole system's security level. In this process, we consider a more practical and stronger threat model in which the cloud server can collude with a small number of data users. To support multiple document search patterns, we construct two AVL trees for the filenames and authors, and a Hierarchical Retrieval Features tree (HRF tree) for the document vectors. A depth-first search algorithm is designed for the HRF tree and the Enhanced Asymmetric Scalar-Product-Preserving Encryption (Enhanced ASPE) algorithm is utilized to encrypt the HRF tree. All the three index trees are linked with each other to efficiently support the search requests with multiple parameters. Theoretical analysis and simulation results illustrate the security and efficiency of the proposed framework.**

***Index Terms*—Cloud computing, privacy-preserving, searchable encryption, document ranked retrieval.**

I. Introduction

CLOUD computing is widely treated as a promising information technique (IT) infrastructure because of its powerful functionalities. It can collect and reorganize huge resources of storage, computing, communication and applications. This makes the cloud users can access the IT services in a flexible, ubiquitous, economic and on-demand manner [1]. Attracted by these excellent properties, more and more data owners tend to outsource their local document management systems to the public cloud. However, an accompanying challenge is how to protect the privacy of sensitive information while maintaining the usability of the uploaded data [2]. Clearly, all these documents need to be encrypted before being outsourced and hence it is severe to design proper mechanisms to realize basic operations on the encrypted document collection. In general, the basic functionalities of a document management system include *INSERT*, *DELETE*, *MODIFY* and *SEARCH*. The first three operations on encrypted database will be discussed in Section VI.C and we now mainly focus on document retrieval mechanism over encrypted cloud files.

Many encrypted document retrieval schemes have been proposed and they can be divided into several categories based on their functionalities, including single keyword Boolean search schemes [3], [4], [5], [6] single keyword ranked search schemes [7], [8], [9], [10], [11] and multi-keyword Boolean search schemes [12], [13], [14], [15], [16], [17], [18], [19], [20]. However, these schemes cannot fully satisfy the data users in terms of document retrieval. In real life, it is extremely common for us to use a set of keywords, such as "searchable", "encryption", "cloud" and "document", to search the interested files in a particular field. Moreover, we hope that the returned results should be sorted in order based on the correlations to the provided keywords. Unfortunately, none of the above schemes can completely meet these requirements.

Recently, privacy-preserving multi-keyword ranked document search schemes have gained extensive attentions of researchers [21], [22], [23], [24], [25]. These schemes make the data users able to retrieve encrypted documents based on a set of keywords and the search processes are similar to that on the plaintext documents from the perspective of data users. Compared with multi-keyword Boolean search, these schemes are more practical and in conformity with the users' retrieval habits. However, these schemes can be further improved in the following aspects.

First, most existing schemes assume that all the data users are trustworthy. This assumption is almost impossible in real life. In fact, the cloud server can easily disguise itself as a data user to wheedle the secret keys out from the data owner with an extremely low cost. Once the cloud server gets the secret keys, all the encrypted documents can be easily decrypted and this is indeed a great blow to the existing schemes. This is the most important cause of designing a novel and practical framework for secure document retrieval in encrypted cloud file systems. Another challenge is that the disguised data users may distribute the decrypted documents to the public. Fortunately, many schemes [26], [27], [28] have been proposed to track the source of file leakage and this can effectively prevent the documents from leaking. Considering that this doesn't fall in the scope of this paper, we ignore this challenge in the rest.

Second, most existing schemes focus on only one type of document retrieval manner and the data users' search experience can be further improved. In reality, the data users may need to search a set of documents by providing filenames, authors, several keywords or any combination of them. Intuitively, we can treat the filenames and authors as common keywords like most existing schemes. However, this manner may decrease the search accuracy. For example, a data user wants to search all the research papers of author "Bob" who is a well-known computer scientist. Clearly, except for Bob's papers, keyword "Bob" also appears in many other papers that reference Bob's work and most of these papers should also contain keyword "computer". Therefore, the data user cannot accurately obtain the interested papers by searching keywords "Bob" and "computer". Another possible method is integrating the multi-keyword Boolean query schemes [12], [13], [14], [15] to the multi-keyword ranked search schemes in [21], [22], [23], [24], [25]. Specifically, the keyword-based Boolean search can first return the candidate documents that contain specific filenames and authors; then, multi-keyword ranked search schemes can rank the candidate documents and return the documents related with the keywords contained in the search request. However, this method is extremely time-consuming considering that the complexity of keyword-based Boolean search is linear to the size of the whole document collection. Therefore, we need to design a totally new framework to satisfy the data users rather than simply combine two types of existing document search schemes.

Third, the search efficiency can be further improved. In multi-keyword ranked document search schemes, a keyword-based index tree is used to search the interested documents. However, it is extremely difficult to design a keyword-based index tree which can perfectly balance the search efficiency and accuracy. Though the

Junsong Fu and Baojiang Cui are with the School of Cyberspace Security and National Engineering Lab for Mobile Network Technologies, Beijing University of Posts and Telecommunications, China. E-mail: {fujs, cuibj}@bupt.edu.cn.

Na Wang is with the School of Cyber Science and Technology, Beihang University, China. E-mail: nawang@buaa.edu.cn.

Bharat K. Bhargava is with the Department of Computer Science, Purdue University, IN 47906, USA. E-mail: bbshail@purdue.edu.

(Corresponding author: Junsong Fu, Na Wang)

This work was supported by National Natural Science Foundation of China (No 62001055, 62102017), Beijing Natural Science Foundation (4204107), Funds of "YinLing" (No. A02B01C03-201902D0).

keyword-balanced binary tree can provide accurate search results, its efficiency is sensitive to the input order of the document vectors [22]; in contrast, the hierarchical-clustering-based index tree provides a better-than-linear search efficiency but result in precision loss [23]. Moreover, searching a keyword tree consumes much more time compared with that of searching a filename tree or an author tree considering that the keyword tree is much more complex.

To improve the security and user experience of encrypted document retrieval system, we consider a stronger threat model in which the cloud server can collude with a small number of data users to collect private information of the documents and index structures. Then, this paper designs a new encrypted document storage and retrieval framework in which a proxy server is employed to act as a bridge between the cloud server and data users. Three index trees including filename tree, author tree and HRF tree, are constructed. In this way, the filenames, authors and common keywords in search requests are of different weights and they are treated differently. The nodes in the three trees are linked with each other based on document identifiers to efficiently support the search requests with multiple parameters. A methodical mechanism is designed to make full use of the information in a query. To support multi-keyword ranked search, the widely used TF-IDF model is employed to model the documents and queries as vectors. Then, the Enhanced Asymmetric Scalar-Product-Preserving Encryption (Enhanced ASPE) algorithm is utilized to encrypt the HRF tree and query vectors while ensuring the accurate relevance score calculation. In addition, a depth-first search algorithm for the HRF tree is also proposed. A theoretical demonstration is provided to illustrate that our scheme can defend against the chosen-plaintext attack model. Meanwhile, simulation result shows that our scheme also greatly outperforms existing schemes in terms of efficiency.

Our contributions are mainly summarized as follows:

- This paper considers the chosen-plaintext attack model which is much stronger than ciphertext-only attack model employed in most existing schemes.
- A new encrypted document storage and retrieval system is designed to improve system security in which a proxy server is employed. The new framework can provide multi-type document search services.
- A complete search mechanism is proposed to improve the search efficiency. Moreover, we propose an updated mechanism for the HRF tree to support dynamic document collection.
- A set of analysis and experiments are conducted to evaluate the performance of the proposed framework in terms of security and efficiency.

The rest of this paper is organized as follows: In Section II, we summarize the related work. Section III states the problem of privacy-preserving multi-keyword ranked search. We present the balanced binary tree and HRF tree in Section IV and V, respectively. The details of the secure document search framework are presented in Section VI. We analyze the security of our framework in Section VII and further evaluate its efficiency in Section VIII. At last, Section IX concludes this paper.

II. Related Work

Cao et al. first propose the privacy-preserving multi-keyword ranked search problem in [21] and they design an initiatory scheme named MRSE. Each document is mapped to a document vector based on term frequencies of the words in the document. A query is transferred to a query vector based on inverse document frequencies of the keywords in the whole document collection. The correlation between a query and a document is then calculated based on the TF-IDF model. The retrieval results of a query are the top-k relevant documents to the query. To protect privacy of documents, both the document vectors and query vectors are encrypted based on secure kNN algorithm [29]. Moreover, a set of strict privacy requirements are established for the following schemes in this field [30]. The disadvantage of MRSE is that all the documents need to be scanned to get the search results of a query and the search efficiency is linear with the cardinality of the document collection. Because the search efficiency of MRSE is low, it cannot be directly used to process extremely large document collections.

To improve document search efficiency of MRSE, two index structures for the encrypted documents are proposed. Xia et al. [22] propose the Keyword Balanced Binary tree (KBB tree) and design a "Greedy Depth-First Search" algorithm for the tree. In KBB tree, each entry in an intermediate node is not smaller than that of all the child nodes. This property performs an important role in pruning redundant searching paths. Moreover, they can dynamically update the index tree with a moderate workload. Though KBB tree greatly improves the search efficiency, the document vectors in the tree are organized chaotically and some redundant paths still need to be visited in document search process. Clearly, the search efficiency can be further improved.

To further optimize the structure of KBB tree, Chen et al. [23] design a novel hierarchical-clustering-based index structure in which the document vectors are organized based on similarities. Specifically, similar document vectors are close with each other in the tree, and vice versa. In this way, it is likely that the retrieval results of a query locate close with each other in the tree and hence most paths in the tree can be pruned in the search process. Simulation results illustrate that the scheme is of a better-than-linear search efficiency. In addition, a verification process is also integrated into their scheme to guarantee the correctness of the results. However, as discussed in [23], this tree cannot guarantee the optimal search accuracy and it is severe to get a balance between search efficiency and accuracy.

Fu et al. [24] assume that the data users cannot select the most proper keywords to search the results. Therefore, they design an interest model for the users to fulfill and revise the provided keywords. Specifically, the interest model of a data user is constructed based on WordNet [31]. However, the document vectors in this paper are constructed based on the whole document collection and hence this structure cannot be dynamically updated. This scheme employs the MDB-tree to improve the search efficiency.

A common vulnerability of the above schemes is that they all employed the ciphertext-only attack model which is a weak threat model in real life. Once the cloud server colludes with a set of data users to conduct the chosen-plaintext attack, the cloud server can recover all the plaintext documents and the vectors. This can be explained by the fact that the data users can access the secret keys.

Recently, some privacy-preserving semantic document search schemes are proposed [32], [33], [34], in which the documents are summarized by the important and simplified sentences rather than the keywords. To our knowledge, semantic document search is a new direction in cloud computing and how to securely share the abstract data for resource-limited data users in cloud computing is discussed in [35]. Moreover, the security problems of existing searchable encryption schemes are discussed in [36], [37], [38], [39]. To define the security clearly, four leakage profile levels are extracted from existing searchable encryption schemes. For different leakage profiles, corresponding attack models are proposed though they mainly focus on single keyword or multi-keyword Boolean search schemes.

III. Problem Statement

A. Notations

- $\mathcal{F}$ – The owner's document collection, denoted as $\mathcal{F} = \{F_1, F_2, \cdots, F_N\}$, which is composed of N file. Each document F_i comprises three parts: *filename*, *authors* and *main body*. The *main body* is treated as a sequence of keywords. Each file has a unique identifier. For convenience, we employ "a document" to represent the "the main body of a document" in the rest.
- $\mathcal{FN}$ – The filename collection of the documents, denoted as $\mathcal{FN} = \{FN_1, FN_2, \cdots, FN_N\}$. Without loss of generality, each

document is assumed to have only one unique filename.

- $\mathcal{AU}$ –The author collection of the documents in $\mathcal{F}$, denoted as $\mathcal{AU} = \{AU_1, AU_2, \cdots, AU_K\}$. We assume that each document can have several different authors and in total K authors exist.
- $\mathcal{C}$ – The encrypted document collection stored in the cloud server, denoted as $\mathcal{C} = \{C_1, C_2, \cdots, C_N\}$. The ciphertexts in $\mathcal{C}$ are obtained by encrypting the files in $\mathcal{F}$ with independent symmetric secret keys $s = \{s_1, s_2, \cdots, s_N\}$, i.e., $\mathcal{C} = e_s(\mathcal{F})$.
- $\mathcal{W}$ – The keyword dictionary with in total m keywords, denoted as $\mathcal{W} = \{w_1, w_2, \cdots, w_m\}$. The dictionary is used to generate the vectors of documents and search requests.
- $\mathcal{I}$ – The encrypted index of $\mathcal{F}$, denoted as $\mathcal{I} = \{I_1, I_2, I_3\}$, where I_1 is the index tree of filenames, I_2 is the index tree of authors and I_3 is the encrypted HRF tree of the main bodies.
- $\mathcal{SR}$ – The search request of a data user, denoted as $\{FN, AU = (AU_1, \cdots, AU_t), MK\}$ where FN is a filename, AU is a set of authors and MK is a set of keywords. Note that, at least one of the three parameters needs to be provided and the default values are set to $null$.
- $\mathcal{TD}$ – The trapdoor of a request $\mathcal{SR}$, denoted as $\mathcal{TD} = \{h_{FN}, (h_{AU_1}, \cdots, h_{AU_t}), E_Q\}$. Specifically, h_{FN}, h_{AU_i} are the corresponding random numbers of FN and AU_i, and E_Q is the encrypted query vector of MK. A trapdoor is the encrypted format of a search request and it can be employed by the cloud server to search the encrypted index $\mathcal{I}$.
- $\mathcal{R}$ – The encrypted result for a search request and it is returned from the cloud server to the proxy server.
- $\mathcal{PR}$ – The plaintext of $\mathcal{R}$ which will be returned to the data users from the proxy server.
- $\mathcal{SK}$ – The pre-set secret keys include two bit-vectors S_1, S_2, and two invertible matrices M_1, M_2.

B. System Model

As shown in Fig. 1, the encrypted document retrieval system involves mainly four entities: *data owner*, *data user*, *proxy server* and *cloud server*.

Data owner has a collection of documents $\mathcal{F} = \{F_1, F_2, \cdots, F_N\}$ and he is responsible for collecting newly generated files. Then, *data owner* outsources the encrypted documents to the *cloud server* with the help of the *proxy server*. Specifically, once a new document is ready to issue, the *data owner* directly sends it to the *proxy server* where the other steps will be completed. Moreover, the data owner can also delete or modify the files in the cloud by sending requests to *proxy server*.

Proxy server is a trusted agency and it links the other three entities. Having received the files from *data owner*, the *proxy server* is responsible for analyzing and encrypting them. An encrypted index structure $\mathcal{I}$ is constructed based on filenames, authors and the document vectors. Both the encrypted index $\mathcal{I}$ and encrypted document collection $\mathcal{C}$ are sent to the *cloud server*. When a query request $\mathcal{SR}$ is received from an authorized *data user*, a trapdoor $\mathcal{TD}$ for $\mathcal{SR}$ will be generated and sent to the *cloud server*. At last, the received search results from the *cloud server* will be decrypted as $\mathcal{PR}$ which is sent to the *data user*. We assume that the *proxy server* can securely communicates with *data owner* and *data users* by symmetric encryption.

Data users are the authorized ones to access the documents. Once a request $\mathcal{SR}$ is sent to the *proxy server*, a set of documents will be received from the *proxy server* and the search process is transparent to the *data user*. In fact, the *data users* do not access any private information of the documents directly such as the secret keys $\mathcal{SK}$ and keyword distributions of $\mathcal{F}$. In this paper, we assume that the filename-based search and author-based search are accurate searches, i.e., the provided filename and authors must be accurate and only the matched documents are returned. In the multiple keywords search, the documents are returned in order according to the relevance scores between the document vectors and query vectors.

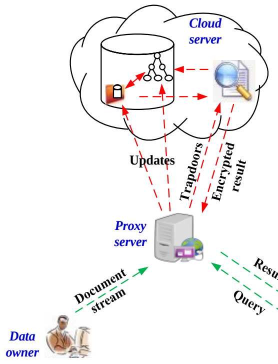


Fig. 1. Encrypted document retrieval system model

The three search patterns are complementary with each other and they provide a better search experience to the *data users*.

Cloud server stores the encrypted documents collection $\mathcal{C}$ and encrypted searchable index $\mathcal{I}$ which are generated by the *proxy server*. Once a trapdoor is received, it needs to search $\mathcal{I}$ and send the search result $\mathcal{R}$, i.e., a set of encrypted documents that match $\mathcal{SR}$, to the *proxy server*. It also needs to update $\mathcal{C}$ and $\mathcal{I}$ in time according to the instructions provided by the *proxy server*.

C. Threat Model

Cloud server model. Similar to the threat models in [21], [22], [23], [24], [25], the cloud server is considered as “honest-but-curious”, which is widely employed in the field of encrypted document retrieval. Specifically, the cloud server properly executes the instructions and however, it is curious to infer and analyze all the received data. We also assume that the cloud sever tries to pretend and bribe data users to get the secret information.

Data user model. In this paper, a small number of data users are assumed to be unreliable and they can leak all their private information to the cloud server. In most existing schemes [21], [22], [23], [24], [25] the authorized data users are assumed to be reliable. They need to hold the secret keys $\{S_1, S_2, M_1, M_2\}$ to generate the trapdoors and hold the symmetric keys $\{s_1, s_2, \cdots, s_N\}$ to decrypt the received results. Note that, the secret keys of all the data users are the same with each other. In this case, if a data user colludes with the cloud server, it is easy to calculate all the document vectors in the index structures based on $\{S_1, S_2, M_1, M_2\}$. Moreover, if $\{s_1, s_2, \cdots, s_N\}$ is also leaked to the cloud server, all the plaintext documents are known to the cloud server. Information leakage problem is an inherent threat to these schemes.

Proxy server model. We assume that the proxy server is controlled by the data owner and it is trusted. The proxy server can properly execute instructions and do not leak its private information to any other entity.

Chosen-plaintext attack model. Considering that the cloud server colludes with a small set of data users, we consider a stronger attack model compared with that in existing schemes [21], [22], [23], [24], [25] i.e., the adversary conducts the chosen-plaintext attack to recover the plaintext documents, filenames, authors and document vectors.

D. Design Goals

Flexibility. The data users can flexibly provide multi-type parameters to search the interested documents, such as a filename, some authors, keywords or any combination of them.

Accuracy. The search results are accurate according to the data users’ search requests and system settings.

Efficiency. The search process achieves logarithmic search efficiency in general and at least sub-linear search efficiency in the worst case.

Dynamicity. The document collection and corresponding index structures can be updated dynamically with a small burden.

Security. In our scheme, we prevent the cloud server from learning the private information about the encrypted document collection. The detailed privacy requirements are summarized as follows:

1) *Document privacy*. The plaintexts of the documents should be strongly protected from the adversaries.
2) *Privacy of FN-AVL tree and AU-AVL tree*. Each node in these two trees represents a filename or an author. Given a node, the corresponding information about filename and author should be protected.
3) *Privacy of HRF tree*. The underlying contextual information of the documents, such as the unencrypted document vectors and the TF, IDF values of keywords, should be protected from the adversaries.

IV. Filename/Author Balanced Binary Search Tree

A. Structure of Filename AVL Tree and Author AVL Tree

Self-balancing binary search trees, such as AVL tree [40], have been widely used to organize data for fast queries. In this paper, we first assign a unique and random number to each filename and author by one-way functions. Without loss of generality, we assume that the filenames, authors cannot be recovered based on the random numbers. Then we build a filename AVL tree called $FN-AVL$ tree and an author AVL tree called $AU-AVL$ tree based on the random numbers to support filename-based search and author-based search. In both of the trees, the left child nodes of a parent node have smaller numbers and the right child nodes have larger numbers. This property significantly improves the efficiency of searching a specific number corresponding to a filename or an author. The time complexities of inserting, deleting and searching a number in the $FN-AVL$ tree are all $O(ln(N))$, where N is the number of the filename, and that in $AU-AVL$ tree are all $O(ln(K))$, where K is the number of authors.

B. Construction of Filename AVL Tree and Author AVL Tree

For a filename FN_i, the corresponding node u in the $FN-AVL$ tree is defined as follows:

$$u = (ID_{FN}\ , func(FN_i), P_{left}, P_{right}), \tag{1}$$

where ID_{FN} is the identifier of the file with FN_i as filename, $func(FN_i)$ is the random number corresponding to the filename, P_{left} and P_{right} are the pointers to the left and right child of node u. The default values of the pointers are set to $null$.

Different from document filenames, each document may have several authors and it is unwise to treat all the authors for a document as an entirety considering that it is very difficult for the data users to accurately provide all the authors of a file. In the $AU-AVL$ tree, each author is treated as an independent entity. For an author AU_i, the node v in the tree is defined as follows:

$$v = (SID_{AU}, func(AU_i), P_{left}, P_{right}), \tag{2}$$

where SID_{AU} is a set of file identifiers with AU_i as an author, $func(AU_i)$ is the random number corresponding to the author AU_i, P_{left} and P_{right} are the pointers to the left and right child of node v. The default values of the pointers are set to $null$. Note that, the number of the nodes in the $AU-AVL$ tree equals to the number of all the authors in the document set. Considering that each document can have several authors and each author can have several files, a file may correspond to several nodes in the $AU-AVL$ tree and each node can also correspond to several files that contain the author represented by the node. When a data user provides several authors, we hope to employ each of them to filter the results collaboratively and hence, in $AU-AVL$ tree, the authors contained in a same file are linked together. In this way, the files with several authors can be easily obtained by intersecting the file sets of author.

Based on the random numbers corresponding to the filenames and authors, we build and update the $FN-AVL$ tree and $AU-AVL$ tree according to the algorithm proposed in [40]. In $FN-AVL$ tree, we employ binary search algorithm to search the tree for a query. To search the files containing a set of authors, we first search the first author and get a set of file candidates. Then, we find the second author through links and update the file candidates. The above process is iterated until all the authors are scanned and get the final search result.

V. Hierarchical Retrieval Feature Tree

A. Document/Query Vectors and Relevance Score Function in Multi-Keyword Document Search

In this paper, the main body of each document is treated as a stream of keywords and we use the normalized TF vector to quantize the documents [41]. The TF value of keyword w_i in F_j is defined as:

$$TF'_{j,w_i} = \ln(1 + f_{j,w_i}), \tag{3}$$

where f_{j,w_i} is the number of times that it appears in F_j. We then normalize TF value of w_i in F_j as follows:

$$TF_{j,w_i} = \frac{TF'_{j,w_i}}{\sqrt{\sum_{w_k \in W}(TF'_{j,w_k})^2}}, \ i = 1,2,\cdots,m. \tag{4}$$

At last the normalized vector of F_j is constructed as follows:

$$V_j = (TF_{j,w_1}, TF_{j,w_2}, \cdots, TF_{j,w_m}). \tag{5}$$

The above constructed document vectors have two advantages. First, the normalized TF vector is a good summary about the content of a document. Second, the normalized TF vector is an inherent property of a document and it is independent of the document collection which may change dynamically. For convenience, we employ the term "document vectors" to represent the "normalized document vectors" in the rest.

As for a query request, consider a game that a data user is interested in a set of documents and he tries to employ a set of keywords MK to describe the documents as clearly as possible. Obviously, he should provide some important keywords with strong capability of locating the interested documents rather than some common words. Therefore, each word needs a weight to reflect its capability, and in this paper, we employ IDF value as the weight of a keyword. The IDF value of w_i is defined as $IDF_{w_i} = \ln(N/N_{w_i})$, where N is the number of documents in the whole document collection and N_{w_i} is the number of documents that contain keyword w_i. Further, the query vector is represented as $V_Q = (q_1, q_2, \cdots, q_m)$ where q_i is 0, if $w_i \notin MK$; and q_i is IDF_{w_i}, if $w_i \in MK$. It can be observed that the IDF value of a keyword is related with the whole document collection and independent of specific documents.

At last, we adopt the widely used "TF-IDF" measurement to calculate the relevance score between a document and a query as follows:

$$RScore(V_j, V_Q) = V_j \cdot V_Q. \tag{6}$$

B. Structure of an HRF Tree

In this paper, we use hierarchical retrieval feature (HRF) tree to organize the document vectors. As shown in Fig. 2, an HRF tree is a height-balanced tree and each node in the tree maps to a cluster of document vectors. Each leaf node is composed of a set of similar document vectors and its HRF vector is extracted from the document vectors. The similar leaf nodes agglomerate with each other to compose the non-leaf nodes until all the document vectors belong to a huge cluster at the root node. Clearly, a higher node in the tree maps to a larger cluster and the root node maps to the cluster composed of all the document vectors.

Two branching factors, B_1, B_2, are employed to control the tree's structure. Specifically, a leaf node L_i contains at most B_1 document vectors and its retrieval vector (RV) is defined as follows:

$$L_i = (HRF, V_1, \cdots, V_k)\ , k \leq B_1 \tag{7}$$

where HRF is the HRF vector of the cluster, V_l is the l-th document vector in the cluster. Each non-leaf node or the root node NL_i contains at most B_2 child nodes and its RV is defined as follows:

$$NL_i = (HRF, HRF_1, child_1, \cdots, HRF_k, child_k), k \leq B_2, \tag{8}$$

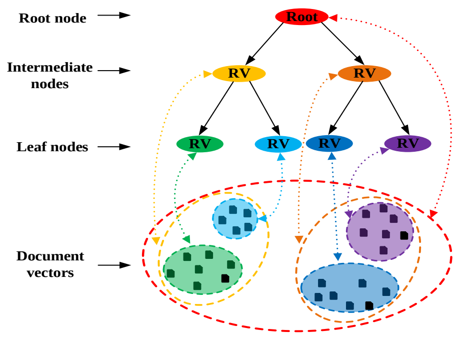


Fig. 2. The structure of an HRF tree

where HRF is the HRF vector of the cluster, HRF_l is the HRF vector of the l-th sub-cluster and $child_l$ is a pointer to the child node corresponding to the sub-cluster.

An HRF vector is a summarization about the corresponding cluster. Given P m-dimensional document vectors, $\{V_j\}$, where $j = 1,2,\cdots,P$, the HRF vector of the cluster is denoted as $HRF = (P, LS, V_{max})$, where $LS = \sum_{j=1}^{P} V_j$, V_{max} is calculated as:

$$V_{max}[i] = max(V_1[i], V_2[i], \cdots, V_P[i]), i = 1,2,\cdots,m. \quad (9)$$

Based on the definition of HRF vector, we can infer that the HRF vectors of the non-leaf nodes and the root node can be calculated based on the HRF vectors of all their child nodes.

Moreover, given an HRF vector, the centroid of a cluster C can be easily calculated as follows:

$$c = LS/P. \quad (10)$$

The relevance score between cluster C and document vector V_j is defined as follows:

$$\text{RScore}(C, V_j) = c \cdot V_j. \quad (11)$$

The relevance score between cluster C and query vector V_Q is defined as follows:

$$\text{RScore}(C, V_Q) = c \cdot V_Q. \quad (12)$$

C. Constructing an HRF Tree

We construct an HRF tree in an incremental manner. The process of inserting V_j into the tree is presented in Algorithm 1. As shown in line 1 to line 6, V_j iteratively descents the HRF tree by choosing the closest child node based on (11) until it reaches a leaf node. After inserting V_j into the leaf node, we update all the infected nodes in a bottom-up manner as shown in line 7 to line 16. In the absence of a split, we simply update the HRF vectors. However, if a leaf node contains more than B_1 document vectors or a non-leaf node contains more than B_2 child nodes, we need to split the node to two new nodes. In this paper, we split a node by choosing the farthest pair of document vectors as seeds, and then redistribute the remaining document vectors based on the closest criteria. A leaf node split requires us to insert a new leaf node to the parent node. In some cases, we may have to split the parent node as well, and so up to the root node. If the root node is split, the tree height increases by one.

As the HRF tree is constructed incrementally, it naturally supports the insert update. However, it is also valuable for the HRF tree to support the delete update. If the data owner wants to delete the document vector of file F_j from the HRF tree, he needs to send the file to the proxy server and then the proxy server is responsible for updating the tree. The detailed process of deleting V_j from the HRF tree is presented as follows:

- *Identifying the document vector*: The proxy server first finds the corresponding number of F_j's filename and then identify the node in $FN - AVL$ tree. Further, V_j can be identified in the HRF tree based on the links between the trees.
- *Modifying the leaf node*: The leaf node L_i containing V_j first deletes the pointer to V_j and then updates its HRF vector. Then L_i scans all the child nodes and if two leaf nodes can combine

Algorithm 1: HRFConstruction (a HRF tree root r, a document vector V_j)

```
1:   $Stack.\text{push}(r)$; $u \leftarrow r$;
2:   while $u$ is not a leaf node
3:     Calculate all the relevance scores between the child nodes of $u$ with $V_j$ based on (11);
4:     $u \leftarrow$ the most relevant child node;
5:     $Stack.\text{push}(u)$;
6:   end while
7:   Insert $V_j$ into $u$;
8:   while $Stack$ is not empty
9:     $u \leftarrow Stack.\text{pop}()$;
10:    if $u$ breaks the limitation of $B_1$ (for a leaf node) or $B_2$ (for a non-leaf node)
11:      Split node $u$ into two nodes and recalculate their HRF vectors;
12:      Update the pointers and the corresponding HRF vectors of the two newly generated
         nodes in the parent node;
13:    else
14:      Update the HRF vector of node $u$ directly;
15:    end if
16:  end while
```

with each other, they are combined to one node. We combine the nodes in order to make the tree compact and this process can be ignored if a small number of vectors are deleted.

D. Searching an HRF Tree

As shown in Algorithm 2, we design a depth-first search algorithm for the HRF tree. After initializing $RList$, the smallest relevance score is used to prune the search paths based on (12). We employ the variable $Stack$ to store the nodes which need to be searched in the future. Once the $Stack$ is empty and all the candidate paths are searched, we can guarantee that the retrieval result is accurate.

In the following, we present the search process in detail and analyze why the structure of the HRF tree can greatly improves the search efficiency. In an HRF tree, the similar document vectors trend to be assigned to the same cluster. Consider a query V_Q and two document vectors V_1 and V_2 where $V_2 = V_1 + V'$, the relevance scores between the query and document vectors are $V_Q \cdot V_1$ and $V_Q \cdot V_2$, respectively. Then the difference between these two relevance scores can be calculated as follows:

$$V_Q \cdot V_1 - V_Q \cdot V_2 = |V_Q \cdot V'| \leq |V_Q||V'|. \quad (13)$$

If V_1 and V_2 are close with each other, $|V'|$ will be small and the relevance scores will be very similar with each other. Consequently, organizing the document vectors based on their similarities can significantly simplify the search process.

We use an example to introduce the simple retrieval process. For a 2-D keyword dictionary, all the document vectors are located on a quarter of a unit circle according to the definition of a document vector. As shown in Fig. 3, the document vectors are divided to 6 clusters $\{a, b, c, d, e, f\}$. A data user generates a query vector and cluster d is the most relevant cluster. Assume that the accurate top-k relevant documents are needed and k is much smaller than the number of document vectors in a leaf node. It is time-consuming to scan all the document vectors in the tree and hence we need to prune the search paths dynamically.

If we can accept an almost accurate result rather than the definitely accurate top-k relevant documents, the search process is extremely easy. Given a query vector V_Q, we first locate the most relevant leaf node in a top-down manner. Specifically, starting from the root node, the query vector V_Q recursively descends the tree by choosing the most relevant cluster according to (12) until the most

Algorithm 2: HRFSearch (an HRF tree root r, a document vector V_Q)

```
1:   Locating the closest leaf node in a similar manner to Algorithm 1;
2:   Initialize $RList$ by selecting the most relevant $k$ document vectors as defined in (12);
3:   $Stack.\text{push}(r)$;
4:   while $Stack$ is not empty
5:     $u \leftarrow Stack.\text{pop}()$;
6:     if the node $u$ is not a leaf node
7:       if $\text{RScore}(V_{u,max}, V_Q) > k\text{thScore}$
8:         Push all the children of $u$ into $Stack$;
9:       else
10:        break;
11:      end if
12:    else
13:      Update $RList$ by calculating the relevance scores between $V_Q$ and the document
         vectors in the leaf node;
14:    end if
15:  end while
16:  return $RList$;
```

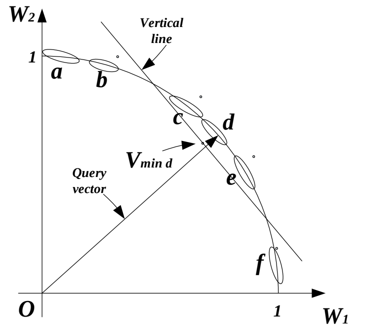


Fig. 3. An example search process with two keywords

relevant leaf node is located. Then, the top-k relevant document vectors in the leaf node are returned as the search result. However, we cannot guarantee that the returned vectors are the accurate result though they are good candidates compared with most other vectors in the tree.

To get the accurate result, we need to further search some nearby clusters. Assume that the relevance score between V_Q and the k-th relevant document vector $V_{d,k}$ in the leaf node d is $V_Q \cdot V_{d,k}$. Another cluster d' should be searched if and only if the maximum relevance score between V_Q and the vectors in cluster d' is larger than $V_Q \cdot V_{d,k}$. In other words, if $V_Q \cdot V_{d',max} \leq V_Q \cdot V_{d,k}$, it is unnecessary to further search cluster d'. Assume that $V_Q \cdot V_{d,k} \geq V_Q \cdot V_{d,min}$, as shown in Fig. 3, only cluster c and e need to be searched, and the other clusters can be ignored. In particular cases, the size of cluster d may be smaller than k and we need to replace it by the second most similar cluster to guarantee the robustness of our scheme. If the document database is large enough, the spatial region of a cluster represented by a leaf node is extremely small and we can prune most of the redundant paths in the tree. We will further evaluate the search efficiency of the HRF tree in Section VIII.

VI. Secure Document Retrieval

A. Linking the Three Retrieval Trees

To efficiently search the documents based on all the parameters in a query, we need to link all the three retrieval trees together. Each node in the $FN-AVL$ tree represents a unique document and each node in the $AU-AVL$ tree represents an author. Consequently, all the nodes in these two trees should be linked to some other nodes in the other trees. However, in the HRF tree, only the elements in leaf nodes correspond to documents directly and only these elements need to be linked with the nodes in the other two trees. Because each document has a unique identifier in the whole document collection, we can link the nodes that contain the same document identifiers in different trees. Once a set of candidate documents are filtered based on one type of search parameters, we can easily access other information about the candidates based on the links between the trees. In this way, we can further refine the search results from the candidates easily and finally get the accurate result. When a tree is updated, the positions of some nodes may change and the information must be delivered to the other trees based on the links to synchronize the link structure.

B. Framework of Privacy-Preserving Document Retrieval

In this section, we present the overall document retrieval framework by mainly employing the functionalities of the proxy server and cloud server.

- $\mathcal{SK} \leftarrow$ **Setup**(): In the initialization phase, the proxy server needs to generate the secret key set $\mathcal{SK}$, including: 1) two randomly generated $(m+m')$-bit vectors S_1 and S_2, and 2) two $(m+m') \times (m+m')$ invertible matrices M_1 and M_2. Note that, S_1 must contains m zeros and m' ones.
- $\mathcal{I} \leftarrow$ **BuildIndex**$(\mathcal{F}, \mathcal{SK})$: For each document, three types of information are first extracted including its filename, all the authors and the main body. We then build the $FN-AVL$ tree, $AU-AVL$ tree and HRF tree. The three index trees need to be linked together based on document identifiers. The first two trees can be directly outsourced to the cloud server, because they store only a set of random numbers rather than the plaintext filenames and authors. In contrast, the HRF tree needs to be encrypted before being outsourced to the cloud. Note that, parameter P in an HRF vector needs not to be encrypted. We treat LS and V_{max} equally to document vectors and encrypt them in the same way. Before encrypting a document vector V_j in the HRF tree, we first extend it to $(m+m')$ dimensions where $m' \geq 0$. Specifically, if $S_{1i}=0$, the i-th dimension of V_j corresponds to a keyword w_r which is extracted from $\mathcal{W}$ in order and $V_j[i]$ is set to TF_{j,w_r}; otherwise, this dimension is an artificial dimension and $V_j[i]$ is set to a random number. Note that the last inserted random number must be a nonzero number and the artificially added dimensions of all the document vectors share the same randomly generated numbers. These rules are related with the structure of the trapdoors and we will discuss it in the following. Further, we spilt each dimension of $V_j[i]$ into $V_j[i]'$ and $V_j[i]''$. Specifically, if $S_{2i}=0$, $V_j[i]'$ and $V_j[i]''$ will be set equal to $V_j[i]$; otherwise, $V_j[i]'$ and $V_j[i]''$ will be set as two random numbers whose sum is equal to $V_j[i]$. We then encrypt V_j as $E_j = \{{M_1}^T V_j{}', {M_2}^T V_j{}''\}$. At last, the outsourced index $\mathcal{I}$ consists of the $FN-AVL$ tree, $AU-AVL$ tree and the encrypted HRF tree.
- $\mathcal{C} \leftarrow$ **EncDocuments**$(\mathcal{F}, s)$: In this paper, the proxy server adopts a secure symmetric encryption algorithm to encrypt the documents in $\mathcal{F}$ based on a set of symmetric secret keys $s=\{s_1, s_2, \cdots, s_N\}$, i.e., $\mathcal{C} = e_s(\mathcal{F})$. Specifically, for each document, a random key of 256 bits is generated. The document identifiers and secret keys are organized in pairwise. Meanwhile, the identifier attribute is set as the main key in the database and hence we can search the secret keys based on the document identifiers through the binary search algorithm. In this way, we can flexibly find the key to encrypt or decrypt a document. Note that, except for the proxy server, all the other entities in the document retrieval system cannot access these keys. At last, the encrypted document collection $\mathcal{C}$ is also outsourced to the cloud server.
- $\mathcal{TD} \leftarrow$ **GenTrapdoor**$(\mathcal{SR}, \mathcal{SK})$: Once a search request $\mathcal{SR}$ is received by the proxy server, it first extracts its parameters including $FN, (AU_1, \cdots, AU_t)$ and MK. For the filename and authors, they are mapped to corresponding numbers by the one-way function $func()$ and we get $h_{FN}, h_{AU_1}, \cdots, h_{AU_t}$. Then, the proxy server constructs query vector V_Q based on MK and $\mathcal{W}$ as discussed in Section V.A. We then extend it to $(m+m')$ dimensions. Specifically, if $S_{1i}=0$, the i-th dimension of V_Q corresponds to a keyword w_r which is extracted from $\mathcal{W}$ in order and $V_Q[i]$ is set to IDF_{w_r}; otherwise, this dimension is an artificial dimension and $V_Q[i]$ is set to a random number. Note that, the value of the last artificial dimension is not a random number and it should be calculated carefully to guarantee that the dot product of the artificially added dimensions in the document vectors and that in V_Q is always 0. Further, we spilt $V_Q[i]$ into $V_Q[i]'$ and $V_Q[i]''$. Specifically, if $S_{2i}=1$, $V_Q[i]'$ and $V_Q[i]''$ will be set equal to $V_Q[i]$; otherwise, $V_Q[i]'$ and $V_Q[i]''$ will be set as two random numbers whose sum is equal to $V_Q[i]$. Finally, we encrypt V_Q as $E_Q = \{{M_1}^{-1} V_Q{}', {M_2}^{-1} V_Q{}''\}$. Clearly, the relevance score of V_j and V_Q can be calculated as:
$$RScore(V_j, V_Q) = V_j \cdot V_Q = E_j \cdot E_Q. \qquad (14)$$
The trapdoor $\mathcal{TD}$ composed of the mapped numbers of the filename and authors, and E_Q are finally sent to the cloud server.
- $\mathcal{R} \leftarrow$ **RSearch**$(\mathcal{TD}, \mathcal{I}, \mathcal{C})$: Three index trees are constructed in our framework and, for different search parameters provided

Algorithm 3: $\mathcal{R} \leftarrow$ **RSearch**$(\mathcal{TD}, \mathcal{I}, \mathcal{C})$

```
1:  if $h_{FN} \neq null$
2:    Search the $FN-AVL$ tree to find the document whose filename correlates with the
      random number $h_{FN}$. We denote the document as $D_1$;
3:    if $(h_{AU_1}, \cdots, h_{AU_t}) \neq null$
4:      Search the $AU-AVL$ tree;
5:      if the random numbers of $D_1$'s authors don't contain the values in $(h_{AU_1}, \cdots, h_{AU_t})$;
6:        The result is empty and return $null$;
7:      else
8:        Return $D_1$ as the result;
9:      end if
10:   else (i.e., $(h_{AU_1}, \cdots, h_{AU_t}) = null$)
11:     Return $D_1$ as the result;
12:   end if
13: else ($h_{FN} = null$)
14:   if $(h_{AU_1}, \cdots, h_{AU_t}) \neq null$
15:     Search the $AU-AVL$ tree to find a set of documents, denoted as $D_2$, that contain all
        the hash values $(h_{AU_1}, \cdots, h_{AU_t})$;
16:     if $E_Q \neq null$
17:       Sort the documents in $D_2$ based on the relevance scores with $E_Q$ and return the
          most relevant top-$k$ documents in order;
18:     else (i.e., $E_Q = null$)
19:       Return all the documents in $D_2$;
20:     end if
21:   else (i.e., $(h_{AU_1}, \cdots, h_{AU_t}) = null$)
22:     if $E_Q \neq null$
23:       Search the HRF tree to find the most relevant $k$ documents with $E_Q$ and return the
          most relevant $k$ documents in order;
24:     else (i.e., $E_Q = null$)
25:       The result is empty and return $null$;
26:     end if
27:   end if
28: end if
```

by the data users, the search process is different. In summary, the filename has the highest importance degree and the keywords have the lowest importance degree. For example, if a query includes a filename and some extra information, we first search the $FN-AVL$ tree to find the legal candidates and then filter the candidates based on the other parameters until the final result is obtained. The detailed process of searching is presented in Algorithm 3. Once the cloud server gets the search result, it extracts the corresponding encrypted documents from the stored document collection $\mathcal{C}$ based on their identifiers. At last, the encrypted documents are sent to the proxy server.

- $\mathcal{PR} \leftarrow$ **DecDocuments**$(\mathcal{R}, s)$: Once the proxy server receives the encrypted search result of a query, it decrypts the encrypted files and finally sends them to the data user.

C. Dynamic Document Collection

In general, three types of update operation, including *INSERT*, *DELETE* and *MODIFY*, should be supported in a database. For brevity, we execute a *MODIFY* request by combining an *INSERT* and a *DELETE* operation and hence only two requests need to be designed. Corresponding operations on the cloud server are inserting and deleting a node to all the three trees, respectively.

We assume that the proxy server stores a copy of the $FN-AVL$ tree, $AU-AVL$ tree and the unencrypted HRF tree locally. We first discuss how to update the HRF tree. Since inserting documents into the collection affects the keyword dictionary $\mathcal{W}$, we need to update the document vectors before updating the structure of the HRF tree. To solve this problem, we preserve some blank entries in $\mathcal{W}$ and set corresponding values in document vectors as 0. If a new keyword is added into $\mathcal{W}$, we just need to replace a blank entry by the new word and then generate new document vectors based on the updated dictionary $\mathcal{W}$.

The update process of the unencrypted HRF tree in the proxy server has been discussed in Section V.C and we need to synchronize the encrypted HRF tree in the cloud server to the unencrypted tree. Specifically, there are three types of update operations: *updating the HRF vector of a node*, *splitting a node* and *combining two nodes*. Correspondingly, three types of update requests are sent to the cloud server from the proxy server. The processes of generating the requests are presented in the following:

- *Generation of an HRF vector update request*: An HRF vector update request for node u in the encrypted tree is defined as $\{u, HRF_{new}\}$, where u is the updated node and HRF_{new} is the new HRF vector of the node. For brevity, the proxy server can put all the HRF vector update requests, caused by an insertion or delete operation, into one message.
- *Generation of a splitting request*: A splitting request for node u is defined as $\{u, u', HRF', p', u'', HRF'', p''\}$, where u is the split node, u', u'' are the new generated nodes, HRF', HRF'' are the HRF vectors of the nodes, and p', p'' are the pointers to the child nodes of u' and u'', respectively.
- *Generation of a combining request*: The request of combining two nodes u', u'' is defined as $\{u', u'', u, HRF_{new}\}$ where u' and u'' are the two combined nodes, u is the new node, HRF_{new} is the HRF vector of u.

Based on the update request, the detailed process of updating the encrypted HRF tree in the cloud server is presented as follows:

- *Updating the HRF vector of a node*: Once an HRF vector update request $\{u, HRF_{new}\}$ is received, the cloud server replaces the original HRF vector of u by HRF_{new}.
- *Splitting a node*: Once a splitting request $\{u, u', HRF', u'', HRF''\}$ is received, the cloud server first finds the parent of u and deletes the pointer to u. Then two new pointers to u' and u'' are inserted to the parent node. In addition, the pointers, p', p'', to the child nodes need to be added into u' and u''.
- *Combining two nodes*: Once a combining request $\{u', u'', u, HRF_{new}\}$ is received, the cloud server first find the parent node of u' and u'', and then delete the pointers to u' and u''. At last, the pointer to u is inserted to the parent node.

We then discuss how to update the two AVL trees. Once the data owner wants to insert a document into the collection, the proxy server needs to send the corresponding numbers of the filename and authors, and the encrypted document to the cloud server. The encrypted document is immediately inserted to the document collection. Then the cloud server inserts a new node into the $FN-AVL$ tree. For each author, the cloud server checks whether it has been inserted to the $AU-AVL$ tree already. If the author already exists in the tree, the identifier of the document is inserted to the author node; otherwise, a new node is inserted into the tree. At last, the links among the three trees are updated. If a document is deleted from the data set, the proxy server needs to send the random numbers of the filename and authors to the cloud server. The cloud server first locates the node in the $FN-AVL$ tree and then deletes the node. Obviously, the structure of the tree also needs to be updated [18]. For an author, the cloud server first locates it in the $AU-AVL$ tree based on its corresponding number, and then deletes the identifier of the document from the node. If the node contains some other identifiers, the node will be kept in the tree, otherwise the node will be deleted. At last, the links between the trees are updated.

VII. Security Analysis

In this paper, the proposed scheme employs several cryptographic algorithms and they are summarized as follows.

- Symmetric encryption algorithms are employed in the process of encrypting documents as discussed in Section III.
- One-way functions are employed when constructing the FN-AVL and AU-AVL trees as discussed in Section IV.
- Enhanced Asymmetric Scalar-Product-Preserving Encryption (Enhanced ASPE) algorithm [29] is employed in the process of constructing the encrypted HRF tree as discussed in Section VI.

We first need to declare that the proposed scheme is built on the above cryptographic algorithms and hence the security of our scheme strongly relies on that of these employed algorithms. Considering that the security proof of these existing cryptographic algorithms doesn't fall in the scope of this paper, for simplicity, two basic assumptions about the employed algorithms are first given as follows:

Assumption 1: In the employed symmetric encryption algorithm, the plaintexts of encrypted documents cannot be recovered without the symmetric secret keys.

Assumption 2: In the employed one-way function, the filenames and authors cannot be recovered given only their mapped random numbers.

In our scheme, the employed symmetric encryption algorithm and one-way function are not strictly restricted to particular types. However, the assumptions are reasonable considering the basic properties of symmetric encryption algorithms and on-way functions [42].

Based on the assumptions provided above, we analyze the security of the proposed scheme. As discussed in Section III.D, we mainly focus on the security of document files and index structures.

A. Document privacy

In existing schemes [21], [22], [23], [24], all the data users preserve the secret keys $\{s_1, s_2, \cdots, s_N\}$ to decrypt the searched documents. In our new threat model, the cloud server can easily recover the plaintexts of encrypted documents once a small number of data users are compromised. However, our framework stores the keys in the proxy server and this properly protects the privacy of documents. In this paper, the ciphertexts of documents are constructed by encrypting the plaintext files with a set of symmetric secret keys. The encryption process is presented as follows:

$$\mathcal{C} = e_s(\mathcal{F}), \tag{15}$$

where $\mathcal{C}$ is the encrypted documents, $\mathcal{F}$ is the plaintext documents, and s is a set of secret keys $\{s_1, s_2, \cdots, s_N\}$. Note that, different documents are encrypted by different and independent keys. Moreover, the secret keys in s are totally controlled by the proxy server and hence the cloud server and data users cannot obtain them.

As discussed in Section III.C, the cloud server can access all the encrypted documents and the data users can access a set of plaintext documents. Considering that a small number of data users may collude with the cloud server, the adversary can easily obtain a set of encrypted documents and the corresponding plaintexts, as shown in the following:

$$InformationLeakage = \{(F_i, C_i), \cdots, (F_j, C_j)\}, \tag{16}$$

where F_i is the plaintext of C_i. Based on Assumption 1, we can infer that they cannot get any information about the plaintexts of other encrypted documents, because the symmetric secret keys are independent with each other. Therefore, document privacy is properly protected in our framework.

Another problem is that the malicious data users may leak a small number of plaintext documents to the public. This phenomenon can be alleviated by tracking the source of file leakage [26], [27], [28] and limiting the number of documents requested by a data user in a period. We do not discuss these techniques in detail here considering that they don't fall in the scope of this paper.

B. Privacy of FN-AVL tree and AU-AVL tree

The $FN - AVL$ tree and $AU - AVL$ tree are stored in the cloud server. By colluding with the data users, the cloud server can get some plaintext filenames and authors, and their corresponding random numbers, as shown in the following:

$$InformationLeakage = \{(H_i, FA_i), \cdots, (H_j, FA_j)\}, \tag{17}$$

where FA_i is a filename or an author, H_i is the random number corresponding to FA_i. Note that, $H_i = func(FA_i)$ and $func()$ is the one-way function employed by the proxy server. Based on Assumption 2, we can infer that the plaintext filenames and authors of other nodes in the tree cannot be recovered even their random numbers are given.

Another challenge is that the adversary may calculate and store the random numbers of the filenames and authors in advance. However, this problem can be alleviated as follows. We can completely cut off the relations between the corresponding numbers to the filenames and authors by randomly selecting a set of numbers for them rather than generating the numbers based on the filenames and authors. Clearly, the cloud server cannot calculate and store the random numbers in advance even the filenames and authors are given. An extra workload for the proxy server is that it needs to maintain a table to retrieve the random numbers for filenames and authors. Considering that many mature data structures have been designed to manage the pairwise data, the workload is an acceptable price for the security of the system. As a consequence, the $FN - AVL$ tree and $AU - AVL$ tree are secure in our scheme.

C. Privacy of HRF tree

In our scheme, the encrypted document vectors stored in the HRF tree are strongly related with the contents of the documents and they are of great importance. In this section, we theoretically prove that the encrypted HRF tree can defend against the chosen-plaintext attack model presented in section III.C. We first describe the challenging game as follows:

Init. The challenger selects a set of documents $\mathcal{F}$ from the data owner and randomly generates a set of secret keys $\mathcal{SK} = \{S_1, S_2, M_1, M_2\}$. Note that, the challenger keeps all the keys secret from the adversary. Without loss of generality, the length of S_1, S_2 is set as m, i.e., no artificial attributes are added. (Note that, the addition of artificial attributes will only increase the security of the tree.)

Setup. The challenger runs **BuildIndex**$(\mathcal{F}, \mathcal{SK})$ algorithm and gives all the encrypted document vectors to the adversary.

Phase 1. The adversary is allowed to issue queries for the encrypted vectors for a set of documents in $P \subseteq \mathcal{F}$.

Challenge. The adversary submits two other documents D_0 and D_1 out of $\mathcal{F}$ to the challenger. The challenger randomly chooses a bit $b \in \{0,1\}$ and maps D_b to a document vector V_b based on $\mathcal{W}$ and S_1. Then, the encrypted vector E_b is constructed based on V_b and the secret keys S_2, M_1, M_2. At last, the challenger sends E_b to the adversary.

Phase 2. Phase 1 is repeated.

Guess. The adversary outputs a guess b' of b.

The advantage of an adversary in this game is defined as $\Pr(b' = b) - \frac{1}{2}$. We say that the privacy of encrypted HRF tree is secure in chosen-plaintext attack model if all polynomial-time adversaries have at most a negligible advantage in the chosen-plaintext attack game.

We theoretically prove the security of HRF tree in Theorem 1.

Theorem 1: For the encrypted HRF tree constructed in this paper, there exists a negligible advantage ϵ for all the polynomial-time adversaries under the assumed system model and threat model presented in Section III such that:

$$\Pr(b' = b) - \frac{1}{2} \le \epsilon. \tag{18}$$

Proof: In our scheme, two steps are needed to get the encrypted document vectors of D_0 and D_1. First, the proxy server needs to map D_0, D_1 to plaintext vectors V_0, V_1 based on the normalized TF model (as discussed in Section V.A) with $S_1, \mathcal{W}$ as secret keys. Without loss of generality, we assume that the entries in the document vectors are constructed based on alphabetical order of keywords. Then, the plaintext vectors V_0, V_1 are mapped to encrypted vectors E_0, E_1 based on secure kNN scheme with S_2, M_1, M_2 as secret keys.

To distinguish E_0, E_1 and output a correct guess b' of b, the adversary first needs to distinguish V_0, V_1 and make the first guess V_0', V_1' of them based on D_0, D_1. For simplicity, we denote $V_0 = f_{tf}(D_0)$ and $V_1 = f_{tf}(D_1)$. The advantage is this process is defined as ϵ_1 and we get:

$$\Pr\left(V_0' = f_{tf}(D_0), V_1' = f_{tf}(D_1)\right) \le \frac{1}{2} + \epsilon_1. \tag{19}$$

Clearly, if we provide $\mathcal{W}$ and S_1 to the adversary, ϵ_1 will be $1/2$. However, the secret information is stored in the proxy server and we can infer that ϵ_1 trends to be smaller than $1/2$. Without loss of generality, we get:

$$\epsilon_1 \leq \frac{1}{2}. \tag{20}$$

Then, in the second guess, the adversary should make full use of query phases. In phase 1 and phase 2 of the game presented previously, we assume that the adversary obtains a set of pairwise relations between documents P and their encrypted vectors E, where $P \subseteq \mathcal{F}$ and $E_i = \textbf{BuildIndex}(P_i, \mathcal{SK})$ for all $P_i \in P$. In this case, the adversary needs to distinguish the two encrypted vectors E_0, E_1 based on the following knowledge:

$$InformationLeakage = \{\mathcal{I}, P, E\}. \tag{21}$$

We can improve the success rate of adversary's guess by extending P to $\{P, V\}$ where V_i is the normalized TF vector of P_i for all $P_i \in P$. For simplicity, we denote $E_0 = f_{knn}(V_0)$ and $E_1 = f_{knn}(V_1)$.

In this paper, we encrypt the vectors based on S_2, M_1, M_2 and the adversary cannot access these secret keys which are stored in the proxy server. Under this situation, the second guess game, i.e., making the guess E_0', E_1' of E_0, E_1 based on V_0, V_1 and the queried information, is strictly the same with the level-3 attack game in [29]. Based on Theorem 6 in [29], we can infer that there exists a negligible advantage ϵ_2 for the polynomial-time adversaries such that:

$$\Pr(E_0' = f_{knn}(V_0), E_1' = f_{knn}(V_1)) \leq \frac{1}{2} + \epsilon_2. \tag{22}$$

By combing (19), (20) and (22), we get:

$$\begin{aligned}\Pr(b' = b) &= \Pr\left(V_0' = f_{tf}(D_0), V_1' = f_{tf}(D_1)\right) * \\ &\quad \Pr(E_0' = f_{knn}(V_0), E_1' = f_{knn}(V_1)) \\ &\leq \left(\frac{1}{2} + \epsilon_1\right) * \left(\frac{1}{2} + \epsilon_2\right) \\ &= \frac{1}{4} + \frac{1}{2}\epsilon_1 + \frac{1}{2}\epsilon_2 + \epsilon_1\epsilon_2 \\ &\leq \frac{1}{2} + \epsilon_2. \end{aligned} \tag{23}$$

Considering that ϵ_2 is a negligible advantage, we prove that equation (18) is always satisfied. Consequently, we claim that the encrypted HRF tree is secure under the proposed threat model. □

In real life, data users search documents based on a set of keywords in most cases and the privacy of trapdoors is also very important. Fortunately, we can prove that the privacy of data users' search requests is also properly protected in our scheme. Considering that the proof process is similar to that of Theorem 1, we don't prove it here.

Based on *Theorem 1*, we can infer that though the cloud server knows the encrypted index structures, the semantic meanings underneath the encrypted vectors cannot be recovered. To further hide the relationships between the trapdoors, in [21], [22], [23], a random factor is added into the trapdoors and hence the relevance scores between query requests and document vectors are also modified. Consequently, the accuracy of the search results decreases which cannot be controlled by the data users. Though the random factors are not employed in our scheme, we will evaluate the search precision of HRF tree with different random factors in Section VIII.C.

VIII. Performance Evaluation

The overall document retrieval efficiency is affected by both the index structures and the time consumptions of executing basic operations. We first theoretically analyze the efficiency of the three proposed index trees in Section VIII.A and then evaluate the overall efficiency of the scheme by experiments in Section VIII.B. At last, the search precision is discussed in Section VIII.C.

A. Efficiency Analysis

The heights of $FN-AVL$ tree and $AU-AVL$ tree are about $\log(N)$ and $\log(K)$, respectively, where N is the number of the documents and K is the number of the authors. As a consequence, the time complexities of inserting, deleting and searching a specific node in the trees are all $O(\log(N))$ and $O(\log(K))$ [40], respectively. Both filename-based and author-based search are accurate searches, i.e., the returned documents contain the filenames or authors, and this search process is equal to search a specific node in the trees.

Different from the above two trees, the structure of the HRF tree is related with the distribution of the document vectors. In the best case, all the leaf nodes contain B_1 children and all the non-leaf nodes contain B_2 children, and hence the depth of the HRF tree is about $\log_{B_1}(N/B_2)$. In this case, the time complexities of inserting, deleting and searching a specific node in the tree are all $O(\log_{B_1}(N/B_2))$. However, if each non-leaf node contains only K_1B_1 $(0 \leq K_1 \leq 1)$ children and each leaf node contains K_2B_2 $(0 \leq K_2 \leq 1)$ child nodes, the depth of the tree will be $\log_{K_1B_1}(N/(K_2B_2))$. In addition, the multi-keyword ranked search process is much complex than searching a specific node in the HRF tree though most of the search paths are pruned. The accurate time complexity is hard to estimate which depends on the distribution of document vectors and the query vector. We will evaluate the search time by experiments in Section VIII.B.

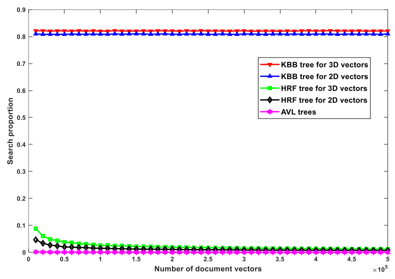


Fig. 4. Search proportion for KBB tree, HRF tree and AVL trees.

To test the efficiencies of the three trees, we compare the trees with the KBB tree proposed in [22] on two and three dimensional spaces, i.e., each document vector is represented by a 2-D or 3-D dimensional vector. We choose the KBB tree as the benchmark because it can return accurate search results similar to the designed trees in this paper. In our simulation, the number of HRF tree's leaf nodes is set to 1000 and the number of documents ranges from 10,000 to 500,000. For each random search query, the top-10 relevant documents are returned. We employ the search proportion measurement to simulate the trees' efficiency and it is calculated by the number of searched document vectors to the number of all the document vectors. We execute the simulation for 100 times and the average simulation results are presented in Fig. 4 and Fig. 5.

It can be observed from Fig. 4 that more than 80% document vectors in KBB need to be searched for both 2-D and 3-D document vectors to obtain the accurate results. This can be explained by the fact that the document vectors in KBB tree are organized chaotically. Consequently, KBB tree cannot lead a query request to the proper position of the tree to obtain the search results efficiently. The HRF tree organizes the vectors based on their similarities. In the search process, the search paths are properly leaded by the HRF vectors of the nodes and most of the search paths are pruned. Compared with the KBB tree, the HRF tree is much more efficient. While the number of document vectors increases from 10,000 to 500,000, the search proportion of the HRF tree decreases from 8.8% to 0.8%. Correspondingly, the search time is significantly shortened. Another interesting observation is that the difference values of the search pro-

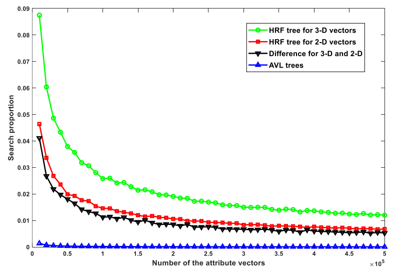


Fig. 5. Search proportion with different numbers of clusters with the same size of the document set, $N = 100{,}000$.

portion for 3-D vectors and 2-D vectors decreases with the increasing of the document vectors' number. We also evaluate the two AVL trees and in fact their search proportion is so small that the search time can be ignored compared with that of the KBB and HRF tree.

Considering that the document vectors are organized as clusters, the number of the leaf nodes also affects the efficiency of the HRF tree and the simulation results are presented in Fig. 5. With the increasing of the number of clusters, the search proportion for both 2-D and 3-D document vectors decreases. The difference values between them also decrease. Except for the search proportion, the size of the tree also greatly affects the search efficiency and to get a balance between them, we adjust B_1, B_2 to make the tree includes about 500 leaf nodes.

Some important properties of the HRF tree are summarized as follows: 1) Searching the "nearly" accurate result is very easy and guaranteeing the strict accuracy of the search results is much more difficult; 2) For a constant number of clusters, the search proportion decreases with the increasing of document vectors' number; 3) The dimension of the document vectors has a large affection on the search proportion and a high dimensional vector space makes a high search proportion.

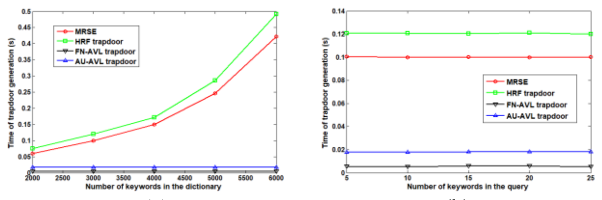
B. *Efficiency Evaluation*

In this section, we evaluate the proposed framework on the Enron Email Data Set [43] and 10,000 records are randomly chosen as our experiment corpus. All the algorithms are implemented on a 2.60 GHZ Intel Core processor, Windows 7 operating system with a RAM of 4 GB. The document retrieval system is mainly composed of four functionalities including constructing the index structure, generating the trapdoor, searching and updating the index structure. The efficiencies of the four functionalities are evaluated respectively in the following.

1) Index Structure Construction

The processes of constructing the two AVL trees are straightforward and include mainly two steps: 1) mapping the filenames and authors to the random numbers, and 2) organizing the numbers by two AVL trees. Clearly, the time costs of building the two AVL trees depend mainly on the numbers of the documents and authors in the document collection. Constructing the encrypted HRF tree mainly includes three phases: 1) mapping the documents to document vectors, 2) building the HRF tree of the document vectors and 3) encrypting the HRF tree. The major computation steps to encrypt a document vector includes a splitting process and two multiplications of a $(m+m')$-dimensional vector and a $(m+m')\times(m+m')$ matrix. To encrypt the whole HRF tree, the total time complexity is $O(N(m+m')^2)$ and hence the time cost for building the encrypted HRF tree mainly depends on the number of documents in the document collection $\mathcal{F}$ and the number of keywords in dictionary $\mathcal{W}$.

Fig. 6(a) shows that the time costs of constructing the HRF tree and the index structure in MRSE are nearly linear with the number of documents. This can be explained by the fact that most time is consumed in the process of constructing the document vectors which is linear to the number of documents. However, compared with the index structure in MRSE, the HRF tree consumes more time because the document vectors are further organized based on their similarities. The two AVL trees are much more time-efficient because they do not need to scan all the words in a document and just need to scan the filename and authors. Fig. 6(b) shows that the time costs of building an HRF tree and the index structure in MRSE are nearly proportional to the number of keywords in the dictionary. The time costs of constructing the AVL trees are independent to the size of keyword dictionary. Though constructing the index structures is of high computational complexity, it is acceptable considering that this is a one-time operation.



(a) (b)

Fig. 7. Time costs of constructing trapdoors. (a) For the different sizes of dictionary with fixed number of query keywords, $t=10$. (b) For the different numbers of query keywords with fixed dictionary, $m=3{,}000$.

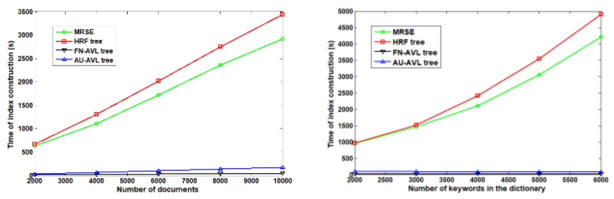
2) Trapdoor Generation

Given a query request including a filename, several authors and t keywords, the generation of a FN trapdoor or a AU trapdoor incurs $O(1)$. Building the HRF trapdoor incurs a vector splitting operation and two multiplications of a $(m+m')$-dimensions vector and a $(m+m')\times(m+m')$ matrix. Consequently, the time complexity is $O((m+m')^2)$ which agrees with the simulation results in Fig. 7(a). The number of keywords in the query has very slight affection on the time costs of generating trapdoors as shown in Fig. 7(b). Building an HRF trapdoor consumes slightly more time than that in MRSE because of the dimension extension.

3) Search Efficiency

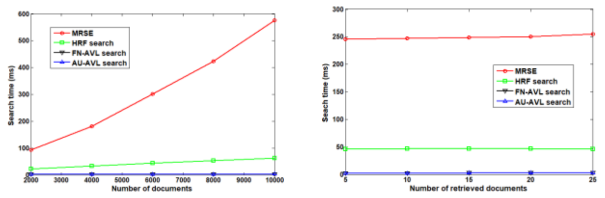
When a data user executes a filename search or authors search, the cloud server needs to execute only $\log(N)$ or $\log(K)$ comparison operations and the time complexities are $O(\log(N))$ and $O(\log(K))$. In a multi-keyword search, the time complexity of computing a relevance score between a trapdoor and a document vector is $O(m+m')$, and the height of the HRF tree is about $\log_{K_1B_1}(N/(K_2B_2))$. Thus, the time complexity of searching a path from the root to the leaf node is $O(\log_{K_1B_1}(n/(K_2B_2))*(m+m'))$. If α percent of all the paths need to be accessed, the upper bound time cost of executing a multi-keyword search is $O(\alpha*(m+m')*(N/(K_2B_2))*\log_{K_1B_1}(N/(K_2B_2)))$.

As shown in Fig. 8(a), in MRSE, all the document vectors need to be scanned to obtain the search result and the time cost is linear to the number of the documents. The search time of HRF tree is much smaller than that of MRSE. The two hash index trees perform much better. Fig. 8(b) presents the search efficiency with the increasing number of retrieved documents. It can be observed that the search



(a) (b)

Fig. 6. Time costs of constructing index structures. (a) For the different sizes of document set with fixed keyword dictionary, $m=3{,}000$. (b) For the different sizes of dictionary with the same document set, $N=5{,}000$.



(a) (b)

Fig. 8. Time cost of executing a query. (a) For the different size of data set with fixed keyword set, $m=3{,}000$. (b) For the different number of retrieved documents with fixed document set and keyword dictionary, $N=5{,}000$, $m=3{,}000$.

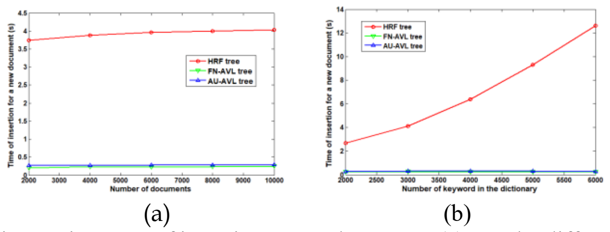


Fig. 9. Time cost of inserting a new document. (a) For the different sizes of document set with fixed keyword dictionary, $m = 3{,}000$. (b) For the different sizes of dictionary with fixed size of document set, $N = 5{,}000$.

time of all the index structures keep relatively stable with the increase of retrieved documents.

4) Update Efficiency

When a document vector is inserted or deleted from the HRF tree, about $O(\log_{K_1 B_1}(N/(K_2 B_2)))$ nodes on the tree need to be updated. Since updating a HRF vector takes $O(1)$ time and the encryption process consumes $(m + m')^2$ time, the overall time complexity of an update operation on the HRF tree is $O((m + m')^2 \log_{K_1 B_1}(N/(K_2 B_2)))$. We illustrate the time cost of executing an update operation through inserting a node into the tree. Note that, we ignore the performance of MRSE in terms of update efficiency in this section considering that no index tree is constructed in that scheme. It can be observed from Fig. 9(a) that when the dictionary is fixed, inserting a document vector into the HRF tree cost nearly logarithmic time with the size of document set. Though the time costs for the AVL trees also increase with an increasing number of document set, it can be ignored compared with that of updating the HRF tree. Fig. 9(b) shows that the update time of the HRF tree is nearly linear to the size of the dictionary with a fixed document set. Similarly, the update time costs of the AVL trees are much smaller than that of the HRF tree. This is reasonable considering that these two trees are much simpler compared with the HRF tree.

C. Search Precision of HRF Tree with Different Random Factors

In our framework, three encrypted index trees are constructed. The search results on the two AVL trees are accurate and hence the search precision is always 100%. In the following, we focus on the search precision of HRF tree with different random factors. Similar to the schemes in [21], [22], [23], a random number can be added to the relevance score between a query vector and a document vector as presented as follows:

$$RScore'(V_j, V_Q) = V_j \cdot V_Q + \delta = E_j \cdot E_Q + \delta, \qquad (24)$$

where δ is randomly selected from a uniform distribution $U(0, b)$. In this way, the search results are slightly different even for the same query request and hence the privacy of access patterns is protected. As shown in Fig. 10, the search accuracy monotonously decreases with the increasing of b. This is reasonable considering that a larger b increases the error of relevance score which misleads the selection process of top- k documents. In conclusion, there exists an interesting tradeoff between search precision and privacy of access patterns. Fortunately, parameter δ can be selected by the data users according to their requirements.

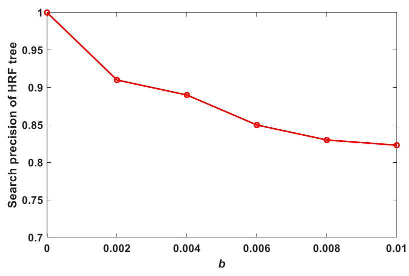


Fig. 10. Search precision of HRF tree with different b.

IX. Conclusion

In this paper, a flexible, secure and efficient privacy-preserving document search framework is proposed based on cloud computing. It supports not only the accurate document search based on filenames and authors, but also the multi-keyword ranked document retrieval. Three tree-based index structures are constructed and an accurate depth-first search algorithm on the HRF tree is designed. When a set of parameters are provided by the data user, the parameters are collaboratively employed to efficiently locate the candidates until the accurate result is finally extracted from the document collection. In our framework, a stronger and more practical threat model is employed in which the cloud server can collude with a small set of data users. Under this assumption, the adversary can execute the chosen-plaintext attack to recover the files, filenames, authors and document vectors. In this case, existing schemes cannot properly protect the privacy of document collection. To defend this new attack, a proxy node is employed in our system to improve the security of the whole system and alleviate the workload of the data owner and data users. Both theoretical analysis and experimental results demonstrate the reliability and efficiency of the proposed framework.

The secure document retrieval framework can be further improved in several aspects. First, the returned top-k relevant documents may not satisfy the data users' requirements and they naturally attempt to obtain the next k relevant documents. Consequently, it is a meaningful future work to design a search scheme that supports dynamic parameter k in the search process. Second, the proxy server is responsible for generating the update information for the HRF tree which is a heavy workload. A better strategy is that the proxy server focuses on security control and the update operations are directly executed by the cloud server. Third, in real life, data users may require more search patterns and some more modules need to be designed and integrated into our framework.

References

[1] K. Ren, C. Wang, and Q. Wang, "Security challenges for the public cloud," *IEEE Internet Computing*, vol. 16, no. 1, pp. 69-73, 2012.

[2] J. Li, W. Yao, Y. Zhang, H. Qian, J. Han. "Flexible and fine-grained attribute-based data storage in cloud computing." *IEEE Trans. on Services Computing*, vol. 10, no. 5, 2016.

[3] D.X.D. Song, D. Wagner, and A. Perrig, "Practical techniques for searches on encrypted data," in Proc. *IEEE Symp. Security and Privacy*, BERKELEY, CA, 2000, pp. 44–55.

[4] E.-J. Goh, "Secure indexes." IACR Cryptology ePrint Archive, vol. 2003, pp. 216, 2003.

[5] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky, "Searchable symmetric encryption: improved definitions and efficient constructions." in *Proc. 13th ACM Conf. Comput. Commun. Security*, Alexandria, Virginia, 2006, pp. 79–88.

[6] S. Jarecki, C. Jutla, H. Krawczyk, M. Rosu, and M. Steiner, "Outsourced symmetric private information retrieval," in *Proc. ACMSIGSAC Conf. Comput. Commun. Secur.*, Nov. 2013, pp. 875–888.

[7] A. Swaminathan, Y. Mao, G. M. Su, H. Gou, A. Varna, S. He, M. Wu, and D. Oard, "Confidentiality-preserving rank-ordered search, " in *Proc. ACM Workshop Storage Security Survivability*, Alexandria, VA, 2007, pp. 7–12.

[8] C. Wang, N. Cao, K. Ren, and W. J. Lou, "Enabling secure and efficient ranked keyword search over outsourced cloud data." *IEEE Trans. Parallel Distrib. Syst.*, vol. 23, no. 8, pp. 1467–1479, Aug. 2012.

[9] S. Zerr, D. Olmedilla, W. Nejdl, and W. Siberski, "Zerber+ r: Top-k retrieval from a confidential index." in *Proc. 12th Int. Conf. Extending Database Technol.: Adv. Database Technol.*, Saint Petersburg, Russia, 2009, pp. 439–449.

[10] C. Wang, N. Cao, J. Li, K. Ren, and W. J. Lou, "Secure ranked keyword search over encrypted cloud data," in *Proc. IEEE 30th Int. Conf. Distrib. Comput. Syst.*, Genova, ITALY, 2010, pp. 253–262.

[11] D. Boneh, G. Di Crescenzo, R. Ostrovsky, and G. Persiano, "Public key encryption with keyword search," in *Proc. Adv. Cryptol-Eurocrypt*, 2004, pp. 506–522.

[12] P. Golle, J. Staddon, and B. Waters, "Secure conjunctive keyword search over encrypted data," in *Proc. Appl. Cryptography Netw. Secur.*, 2004, pp. 31–45.

[13] D. Boneh and B. Waters, "Conjunctive, subset, and range queries on encrypted data," in *Proc. 4th Conf. Theory Cryptography*, 2007, pp. 535–554.

[14] A. Lewko, T. Okamoto, A. Sahai, K. Takashima, and B. Waters, "Fully secure functional encryption: Attribute-based encryption and (hierarchical) inner product encryption," in *Proc. 29th Annu. Int. Conf. Theory Appl. Cryptographic Tech.*, 2010, pp. 62–91.

[15] D. Cash, S. Jarecki, C. Jutla, H. Krawczyk, M.-C. Rosu, and M. Steiner, "Highly-Scalable Searchable Symmetric Encryption with Support for Boolean Queries," in *Proc. Adv. Cryptol*, 2013, pp. 353–373.

[16] L. Ballard, S. Kamara, and F. Monrose, "Achieving efficient conjunctive keyword searches over encrypted data," in *Proc. 7th Int. Conf. Inf. Commun. Secur.*, 2005, pp. 414–426.

[17] Y. H. Hwang and P. J. Lee, "Public key encryption with conjunctive keyword search and its extension to a multi-user system," In *Proc. 1st Int. Conf. Pairing-Based Cryptography*, 2007, pp. 2–22.

[18] B. Zhang and F. Zhang, "An efficient public key encryption with conjunctive-subset keywords search," *J. Netw. Comput. Appl.*, vol. 34, no. 1, pp. 262–267, 2011.

[19] E. Shen, E. Shi, and B. Waters, "Predicate privacy in encryption systems," in *Proc. 6th Theory of Cryptography Conf. Theory Cryptography.*, 2009, pp. 457–473.

[20] J. Katz, A. Sahai, and B. Waters, "Predicate encryption supporting disjunctions, polynomial equations, and inner products," in *Proc. Adv. Cryptol.*, 2008, pp. 146–162.

[21] N. Cao, C. Wang, M. Li, K. Ren, W. Lou, "Privacy-preserving multi-keyword ranked search over encrypted cloud data," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 1, pp. 222-233, 2014.

[22] Z. Xia, X. Wang, X. Sun, Q. Wang. "A secure and dynamic multi-keyword ranked search scheme over encrypted cloud data," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 2, pp. 340-352, 2016.

[23] C. Chen, X. Zhu, P. Shen, J. Hu, S. Guo, Z. Tari and A.Y. Zomaya. "An efficient privacy-preserving ranked keyword search method," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 4, pp. 951-963, 2016.

[24] Z. Fu, K. Ren, J. Shu, X. Sun, F. Huang. "Enabling personalized search over encrypted outsourced data with efficiency improvement," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 9, pp. 2546-2559, 2016.

[25] W. Sun, B. Wang, N. Cao, M. Li, W. Lou, Y. T. Hou, and H. Li, "Privacy-preserving multi-keyword text search in the cloud supporting similarity-based ranking," in *Proc. ACM 8th SIGSAC Symp. Inf., Comput. Commun. Security*, 2013, pp. 71–82.

[26] Z. Jalil and A.M. Mirza, "A review of digital watermarking techniques for text documents," in *Proc. Int. Conf. Inf. Multimedia Technol.*, 2009, pp. 230-234.

[27] H. Fang, W. Zhang, Z. Ma, H. Zhou, S. Sun, H. Cui, "A Camera Shooting Resilient Watermarking Scheme for Underpainting Documents," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 30, no. 11, pp. 4075-4089, Nov. 2020.

[28] Z. Ni, Y.Q. Shi, N. Ansari, W. Su, "Reversible data hiding," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 16, no. 3, pp. 354-362, 2006.

[29] W. K. Wong, D. W. Cheung, B. Kao, and N. Mamoulis, "Secure knn computation on encrypted databases," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2009, pp. 139–152.

[30] Y.-C. Chang and M. Mitzenmacher, "Privacy preserving keyword searches on remote encrypted data," in *Proc. 3rd Int. Conf. Appl. Cryptography Netw. Security*, 2005, pp. 442–455.

[31] G. A. Miller, "WordNet: a lexical database for English," *Commun. ACM*, vol. 2, no. 11, pp. 39–41, 1995.

[32] Z. Fu, X. Wu, Q. Wang, K. Ren. "Enabling Central Keyword-based Semantic Extension Search over Encrypted Outsourced Data," *IEEE Transactions on Information Forensics and Security*, vol. 12, no. 12, pp.2986-2997, 2017.

[33] Z. Fu, F. Huang, K. Ren, J. Weng, C. Wang, "Privacy-preserving Smart Semantic Search based on Conceptual Graphs over Encrypted Outsourced Data," *IEEE Transactions on Information Forensics and Security*, vol. 12, no. 8, pp. 1874-1884, 2017.

[34] Z. Fu, F. Huang, X. Sun, A.V. Vasilakos, C.N. Yang, "Enabling Semantic Search based on Conceptual Graphs over Encrypted Outsourced Data," *IEEE Transactions on Services Computing*, vol. 12, no.5, pp. 813-823, 2019.

[35] J. Li, Y. Zhang, X. Chen, Y. Xiang, "Secure attribute-based data sharing for resource-limited users in cloud computing," *Computers and Security*, vol. 72, pp. 1-12, 2018.

[36] D. Cash, P. Grubbs, J. Perry, T. Ristenpart, "Leakage-abuse attacks against searchable encryption," in *Proc. of the 22nd ACM SIGSAC conference on computer and communications security*, 2015, pp: 668-679.

[37] M. Naveed, S. Kamara, C.V. Wright, "Inference attacks on property-preserving encrypted databases," in *Proc. of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, 2015, pp: 644-655.

[38] R. Bost, B. Minaud, O. Ohrimenko, "Forward and backward private searchable encryption from constrained cryptographic primitives," in *Proc. of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017, pp: 1465-1482.

[39] Y. Zhang, J. Katz, C. Papamanthou, "All your queries are belong to us: The power of file-injection attacks on searchable encryption," in *Proc. of the 25th USENIX Security Symposium*, 2016, pp: 707-720.

[40] G. Adelson-Velsky, L. Evgenii Landis, "An algorithm for the organization of information," *in Proc. of the USSR Academy of Sciences (in Russian)*, 1962, pp. 263–266. English translation by M.J. Ricci in Soviet Math. Doklady, vol. 3, pp. 1259–1263, 1962

[41] C. D. Manning, P. Raghavan, and H. Schutze, "Introduction to information retrieval," Cambridge, U.K.: Cambridge Univ. Press, 2008.

[42] H. Delfs and H. Knebl, "Introduction to Cryptography: Principles and Applications," New York, NY, USA: Springer, 2007.

[43] W.W. Cohen, "Enron Email Data Set," https://www.cs.cmu.edu/~./enron/, 2015.

Junsong Fu received his Ph.D. degree in communication and information system from Beijing Jiaotong University in 2018. He is now serving as an assistant professor in the school of cyberspace security in Beijing University of Posts and Telecommunications. His research interests include in-network data processing, network security and information privacy issues in distributed systems and Internet of Things.

Na Wang received her Ph.D. degree from the School of Mathematical Sciences at Xiamen University in 2018, now an assistant professor in the School of Cyber Science and Technology in Beihang University. Her research interests include cryptography, message sharing and information security issues in distributed systems and cloud systems.

Baojiang Cui received the BS degree from the Hebei University of Technology, China, in 1994, the MS degree from the Harbin Institute of Technology, China, in 1998, and the PhD degree in control theory and control engineering from Naikai University, China, in 2004. He is currently a professor at the School of Computer Science, Beijing University of Posts and Telecommunications, China. His main research interests include detection of software, cloud computing and the Internet of Things.

Bharat K. Bhargava (F'93) is a Professor of Computer Science at Purdue University. Prof. Bhargava is the founder of the *IEEE Symposium on Reliable and Distributed Systems*, *IEEE conference on Digital Library,* and the *ACM Conference on Information and Knowledge Management*. He is the editor-in-chief of four journals and serves on over ten editorial boards of international journals. Prof. Bhargava has published hundreds of research papers and has won five best paper awards in addition to the technical achievement award and golden core award from IEEE. He is a life fellow of IEEE.