Research Article

Secure and Efficient Semantic Extension Search over Encrypted Documents by Integrating Cloud and Fog Systems

Na Wang, J. Jian Jiao, Shancheng Zhang, and Junsong Fu

1 School of Cyber Science and Technology, Beihang University, Beijing 100191, China
2 School of Cyberspace Security, Beijing University of Posts and Telecommunications, Beijing 100876, China

Correspondence should be addressed to Junsong Fu; fujs@bupt.edu.cn

Received 17 November 2021; Accepted 23 June 2022; Published 9 July 2022

Abstract

With the explosive increase of document files, more and more data owners outsource their documents to the public cloud, which can decrease the costs of local data management systems. However, the problem of information privacy leakage in the cloud is a great challenge and it has been attracting more and more attention. In this article, we propose a secure and efficient document search scheme, named SES, based on both the cloud and fog systems. All the documents are symmetrically encrypted before being outsourced to the cloud, and an index vector is constructed based on the keywords for each document. Specifically, we integrate the position information of keywords into the TF-IDF model to generate document vectors, which are accurate and inherent summarizations about the documents. In query requests, a data user needs to provide a set of keywords, which are first extended by the Word2Vec tool and then mapped to a query vector. The extension process of keywords makes the provided keywords more comprehensive and accurate, and hence, it improves document search accuracy. To achieve the forward and backward security, both the document vectors and query vectors are appended with an ingenious vector. The relevance score between a document and a query is defined as the inner product of the document vector and the query vector. We return the most \( k \) relevant documents as the search results to the data users. To protect the contextual information stored in the document and query vectors, we encrypt the vectors by the secure kNN algorithm. To improve the search efficiency, a searchable index structure for the document set is constructed based on the Diffie–Hellman secret key negotiation algorithm. The analysis and simulation results illustrate that the proposed scheme performs well in terms of both security and search efficiency.

1. Introduction

With the rapid development of big data and cloud computing technologies, more and more data owners choose to outsource the data to the cloud, in order to reduce the cost of local data maintenance, and share the data among a large set of data users in a more convenient manner. However, outsourcing data to the cloud server will bring security risks to data privacy, because the cloud server is an “honest and curious” semi-trusted entity. In real life, the cloud server may attempt to mine the privacy of documents and query requests, or leak the privacy information of documents to the unauthorized data users, which are huge losses for the data owners. Therefore, the data need to be encrypted before being outsourced to the cloud server, and this can properly protect the data privacy. However, an accompanying challenge is that the encrypted data uploaded to the cloud server cannot be retrieved in plaintext manner and the search schemes in the traditional environment are no longer feasible. It is severely necessary to design an effective ciphertext retrieval scheme, which is suitable for the cloud computing environment.

Searchable encryption has been widely researched, and existing schemes are mainly divided into two categories, that is, symmetric searchable encryption and asymmetric searchable encryption [1]. In the initial phase, most existing symmetric searchable encryption schemes only supported single keyword search [2–6]. However, in the actual situation, in order to describe the content of interest more accurately, users often input multiple keywords for interested
document retrieval and hope that the returned search results are the most related \( k \) files to the input keywords. In order to meet this demand, in recent years, many schemes supporting multikeyword sorting and retrieval have been proposed.

Cao et al. [7] first proposed a privacy-preserving multikeyword ciphertext retrieval result ranking scheme (MRSE) in the cloud computing environment to efficiently retrieve the ciphertexts of documents based on a set of keywords. They adopted the vector space model to calculate the inner product of document vectors and retrieval vectors, and ranked the document retrieval results based on the results of the inner product operation. After that, a set of schemes [8–10] are proposed in this research branch. Although they share a similar framework, the search efficiency, flexibility, and accuracy in the encrypted document search process are improved. However, most existing schemes need to be further improved in the following several aspects.

First, most existing schemes did not thoroughly consider the influence of positions on the importance degrees of keywords in a document. The same keyword in the title, index terms, conclusion, and other parts of a document is assumed to have the same importance. This is unreasonable considering that the keywords in the title and index terms are more important than that in other parts. Apparently, the authors always carefully design the title of a document to emphasize its field and diversify the document to the other documents in the same field. Therefore, the keywords in the title are more important and we can infer that the position of a keyword is also an inherent property.

Second, the forward and backward security problem in the encrypted document retrieval field is proposed recently [11]. In dynamic document collection, the cloud server can employ the forward and backward attack to analyze the contextual information of the encrypted documents, document vectors, and trapdoors. Specifically, the cloud server may employ the outdated trapdoors to search the newly inserted documents or search the deleted documents by the new trapdoors. Considering that this threat is very novel, most existing schemes cannot properly defend it. Apparently, it is severe to improve the security of the existing document search systems.

Third, in some existing schemes [7], the encrypted document vectors are randomly organized and the search efficiency is very low considering that all the vectors need to be scanned in a query. To improve efficiency, the vectors are embedded into an index tree in some other schemes [12, 13]. With the increase in the size of the document collection, the index tree also monotonously increases and the document search time also increases. If we can prune a set of documents based on the query request before document search, the search efficiency can be further improved.

In this article, we propose a novel scheme to support secure and efficient document search over encrypted cloud file systems. First, we map the documents to a set of index vectors based on a revised TF-IDF model in which the positions of keywords are taken into consideration. Then, an index structure, named search table, for these document vectors is designed to improve search efficiency. In the document retrieval phase, the data users provide a set of keys and they are extended by the Word2Vec tool. Then, the extended keyword set is mapped to a query vector. To properly protect the contextual information in document vectors and query vectors, they are encrypted by the secure kNN algorithm [14] before being uploaded to the cloud server. Although the cloud server does not know the plaintexts of all the vectors, it can accurately calculate the relevance scores between a query vector and all the document vectors. At last, the search results are returned to the data users and the document search process is completed. Benefiting from the encrypted index structure designed based on the Diffie–Hellman secret key negotiation algorithm, the search complexity is shortened to logarithmic complexity.

Our contribution can be mainly summarized as follows:

(i) We set different position weights for the keywords in different positions and design a method to calculate the overall position weight scores. Then, we calculate the correlation scores between the keywords and documents by combining both the domain-weighted scoring method and \( \text{TF-IDF} \) model.

(ii) We extend the search keyword set, which is provided by an authorized data user, by applying the Word2Vec tool. The extended keyword set is more comprehensive, and the keywords can express the semantic information of the data user in a more accurate manner.

(iii) The document index vectors and query vectors are encrypted based on the secure \( k \)-nearest neighbor algorithm. Moreover, an extra vector is also appended after these vectors. In this way, the document collection in our scheme can be dynamically updated with forward security and backward security.

(iv) A searchable encrypted index table is designed by embedding the Diffie–Hellman key agreement scheme. With the exponential growth of the number of documents, the search time is shortened from exponential time complexity to linear time complexity.

The rest of this article is organized as follows. Section 2 summarizes the related work in the field of encrypted document search. The system model, threat model, and design goals are discussed in Section 3. We present the proposed scheme SES in Section 4. The update problem of document set is discussed in Section 5. Section 6 analyzes the security of our scheme, and the efficiency is discussed in Section 7. At last, we conclude this article in Section 8.

2. Related Work

In the multikeyword ranked document search scheme proposed by Cao et al. [7], the document vectors are randomly organized and the search complexity linearly increases with the increase in the document number. In a query process, all the relevance scores between the query vector and document vectors need to be calculated and
ranked to get the search results. Apparently, the retrieval efficiency needs to be improved.

Xia et al. [15] designed a novel index structure named keyword balanced binary tree (KBB tree) to manage the encrypted document vectors. The most important property of the tree is that the relevance score between a node and a query vector is always not less than that between the child nodes and the query. This property is used to dynamically prune the search paths, and the search efficiency greatly improves.

A disadvantage of the KBB tree is that it is large in size, because the vectors are chaotically located in different clusters. The similar document vectors may belong to different clusters, and the vectors in a cluster may vary different from each other. Chen et al. [16] design an index structure based on the hierarchical clustering algorithm. Similar document vectors gradually aggregate with each other to form larger clusters until all the vectors belong to a single cluster. Although this index structure can provide near-optimal results, it cannot support completely accurate document retrieval services. In Ref. [17], the documents are encrypted while maintaining the orders and then the encrypted documents are searched by multiple keywords. The simulation results show that the search accuracy is improved.

Some other tree-based index structures are also proposed in this field. In Ref. [18], B+ tree is employed to support dynamic document retrieval scheme. In Ref. [19], the document index is constructed based on the binary tree and it greatly improves the search efficiency while decreasing the cost of storage space. To support dynamic document sets, Sun et al. [12] propose a novel multiple keywords document search based on the balanced binary tree. To further decrease the cost of index structure and time consumption, Feng et al. [20] propose a distributed document retrieval model based on the Boolean filter.

Except for retrieval accuracy and efficiency, the scalability of encrypted document search is also an important measurement. In Refs. [21–26], the authors research on the algorithms of dynamic inserting, updating, and deleting on the cloud server. Several search schemes are proposed to support multiple keyword retrieval. However, in the search process, the data users must provide the accurate query keywords and the documents containing the keywords are returned. The relevance between keywords are not considered. Apparently, this service is quite simple and it cannot satisfy the data users in real life. In fact, when the data users are not familiar with a specific region, it is extremely difficult for them to always provide a set of proper keywords. In this case, the above schemes fail. Therefore, when the provided keywords cannot fully reflect the intentions of data users, the cloud server needs to return the results based on the semantic meanings of the keywords.

To return proper search results even the keywords provided by the data users are impertinent, the semantic information and relations of the keywords need to be taken into consideration [27–30]. Xia et al. [13, 28] employ the semantic meanings of query keywords in document retrieval. They first construct a semantic graph of keywords for the data users. In the query process, the query keywords are properly rectified to a new query and it is used to search the document collection. This scheme also supports multiple keywords search. In Refs. [9, 31–34], multikeyword-based fuzzy search schemes are proposed and they can return the best fit results.

Although many encrypted document retrieval schemes have been proposed, most existing schemes ignore the impact of positions on the keywords’ importance degrees. In these schemes, all the keywords in a document are equally treated no matter where they are located and as a consequence, the search accuracy can be improved. Similar to our scheme, Sun et al. [35] improve the scheme in Ref. [7] by assigning different weights to different keywords when constructing document vectors and query vectors. The relevance scores are calculated by cosine, and search accuracy increases. However, a disadvantage of the two schemes is that both the document vectors and query vectors are constant and they do not support dynamic document collection.

Moreover, all the existing schemes did not consider the forward and backward security and they need to be improved in terms of security. Wei et al. [11] first proposed the definitions of the forward security and backward security. They assume that the cloud server is untrusted and it may furtively store the historical query vectors and deleted documents. Then, the cloud server can search the new documents based on historical query vectors or retrieve the deleted documents based on the new query vectors. Apparently, this may leak some private information about the encrypted document collection and queries.

3. Problem Statement

3.1. System Model. In order to meet the requirements of both efficiency and security in the encrypted document retrieval process, this article uses the hybrid cloud-fog system model shown in Figure 1. The whole system mainly comprises four entities, including the data owner, data user, fog node, and semi-trusted public cloud server. The main functions of these entities are presented in the following.

Data owner is responsible for collecting document files. For a collected document set $\mathcal{F}$, he first extracts $m$ keywords from the plaintext document set and they are used to form a keyword dictionary $\mathcal{W}$ of the document set. He then sets the weights of the keywords in $\mathcal{W}$ at different positions of the document and calculates the correlation score between the keywords and the documents based on the modified TF-IDF model. After that, each document $F_i$ is mapped to document index vector $D_i$ based on the keyword dictionary. The value of a keyword in the corresponding position of the index vector $D_i$ is set as the product of position weight score and correlation score. The ciphertext set $\mathcal{C}$ and the searchable encrypted index $\mathcal{I}$ are obtained by encrypting $\mathcal{F}$ and $\mathcal{D}$, respectively. Finally, the data owner sends $\mathcal{W}$ to the fog node. Meanwhile, the ciphertexts of documents $\mathcal{C}$ and the search table composed of $\mathcal{I}$ are uploaded to the cloud server.

An authorized data user can search his interested documents with the help of the document retrieval system.
Specifically, he needs to construct an enquiry request $Q^*$ based on a set of interested keywords and then sends it to the fog node. After a large amount of calculation in the fog node and cloud node, the ciphertexts of the ranked top-$k$-related encrypted documents will be returned to the data user. At last, the plaintext target document can be decrypted by the data user based on the symmetric secret key distributed by the data owner.

**Fog node** is an “honest and reliable” entity with a high security level and a small storage space [36–38]. This is reasonable considering that the fog nodes are usually controlled by the data owner. The fog node needs to receive and store the keyword dictionary $W$ uploaded by the data owner. Given an enquiry request $Q^*$ sent by the user, the fog node uses Word2Vec to extend the semantics of $Q^*$ by adding some extra related keywords into the query. Once the fog node obtains the semantic extended enquiry request $\bar{Q}$, it judges whether the keywords in $\bar{Q}$ match the keywords in the corresponding positions in $W$ and gets the search vector. To satisfy forward and backward security, the fog node expands $\bar{Q}$ by appending an extra vector to obtain $\bar{Q}$. At last, the fog node encrypts $\bar{Q}$ to get the document retrieval trapdoor $T_{\bar{Q}}$ based on secure kNN algorithm and sends it to the cloud server.

**Cloud server** is an “honest and curious” semi-trusted entity with a low security degree and a large storage space. It can be used to store ciphertext document sets and a search table made up of encrypted indexes. The cloud server uses the received document retrieval trapdoor $T_{\bar{Q}}$ and encryption index $I_j$ to get a set of candidates. To accurately locate the search results, the cloud server calculates the inner products of the trapdoor and candidate vectors. At last, the cloud server sorts the candidate ciphertext documents according to the results of inner product operation and returns the ciphertexts of the most $k$ relevant documents to the data user.

### 3.2. Threat Model

The public cloud server is generally regarded as an “honest and curious” semi-trusted entity. In general, it honestly executes the instructions of the data owner, fog node, and data users, and it does not disclose the privacy information of documents to the public. However, we assume that it is “curious” to analyze and mine users’ retrieval requests, and obtain the privacy information of data users’ requests. Note that, the data users are not willing to disclose their search requests, because this may leak some information about their behaviors. The cloud server also tries to explore the contextual information stored in the ciphertexts of documents, encrypted index vectors, and trapdoors. This contextual information may leak the contents of the documents and query requests. Based on the effective information obtained by the public cloud server in the system, this article mainly considers the following three threat models.

#### Known ciphertext threat model
In this article, the public cloud server can obtain the ciphertext documents submitted by the data owner. However, it cannot obtain any effective plaintext information about the documents. In this case, the public cloud server can only choose the ciphertext only attack mode.

#### Known background threat model
The public cloud server can statistically analyze the hidden information in the user’s retrieval records according to the user’s retrieval requests and can mine some other useful information, such as the user’s document retrieval preference, associated records, and retrieval results. Therefore, the public cloud server can carry out statistical analysis-based attacks.

#### Forward and backward threat model
In this model, the cloud server is curious and it may store all the deleted document index and outdated trapdoors. This information can be used to analyze the private information...
of newly generated documents and trapdoors. Therefore, the proposed scheme should be forward and backward secure. Specifically, the cloud server cannot retrieve the deleted documents by the new-generated trapdoors and we call this the backward security. Meanwhile, the old-generated trapdoors cannot be employed to the new-inserted documents and this is the forward security.

3.3. Design Goals

(i) Search efficiency. In order to cope with the explosive growth of the number of documents in the big data era, we have to design a novel encrypted index structure to improve the retrieval efficiency of documents.

(ii) Query accuracy. Query accuracy is a basic requirement for the data users, and they always want to access their interested content as accurate as possible. The proposed scheme should support query expansion before submission, so as to solve the limitation of improper keywords provided by the data users.

(iii) Privacy-preserving. Our goal is to prevent cloud servers from obtaining additional sensitive information about the document sets, indexes, and search requests. Specifically, symmetric cryptography is a method to protect document privacy. We use the secure kNN algorithm to encrypt the index and search request, which can prevent the adversary from stealing useful information.

(iv) Forward and backward security. The document collection is dynamic, and the documents can be either deleted from the collection or inserted to the collection. The proposed scheme should be forward and backward secure. Specifically, employing the outdated trapdoor to search the newly inserted documents or searching the deleted documents by the new trapdoors is illegal.

4. Proposed Scheme

In this section, we define the framework of secure and efficient document Search (SES) scheme. First, we introduce the notations in this article and the way of constructing the document vectors. Then, we introduce the Word2Vec tool to extend the query. At last, we present the details of the whole scheme.

4.1. Notations

(i) $\mathcal{F}$—the plaintext document collection, denoted as a set of $n$ document files $\mathcal{F} = (F_1, F_2, \ldots, F_n)$.

(ii) $\mathcal{C}$—the encrypted document collection, denoted as $\mathcal{C} = (C_1, C_2, \ldots, C_n)$. In this article, each document $F_i$ is encrypted to $C_i$ based on a symmetric encryption algorithm with secret key $s_i$. The employed secret keys for different documents are different and independent with each other.

(iii) $\mathcal{W}$—the keyword dictionary, that is, the keyword set consisting of $m$ keywords selected from the document collection by the data owner, denoted as $\mathcal{W} = (w_1, w_2, \ldots, w_m)$.

(iv) $\mathcal{D}$—the document index set, denoted as $\mathcal{D} = (D_1, D_2, \ldots, D_n)$, where $D_i$ is constructed based on $F_i (i = 1, 2, \ldots, n)$.

(v) $\mathcal{Q}$—the vector generated by the dimension extension of $\mathcal{D}$. The extension of $\mathcal{D}$ is employed to match the extension process of $Q$.

(vi) $\mathcal{F}$—the searchable encrypted index generated from $\mathcal{D}$.

(vii) $Q^*$—a subset of keywords in $\mathcal{W}$, representing the enquiry request provided by a data user.

(viii) $\mathcal{Q}'$—the keyword set formed by the semantic extension of $Q$. Note that, the extra keywords are also selected from $\mathcal{W}$ and hence $\mathcal{Q}' \subseteq \mathcal{W}$.

(ix) $Q$—the search vector constructed based on $\mathcal{Q}'$, denoted as $Q = [Q[1], Q[2], \ldots, Q[m]]$. Note that $Q[i]$ can be either 0 or 1.

(x) $\mathcal{Q}$—the vector generated by the dimension extension of $Q$. This extension process is employed to defend the forward and backward security of the system.

(xi) $T_Q$—the encrypted form of $\mathcal{Q}$. In this article, we name $T_Q$ as the trapdoor of the search request. The trapdoor can be used to search the encrypted documents by the cloud server; however, the plaintext of $T_Q$ cannot be recovered without the secret key.

(xii) $L$—a collection of numbers initialized to $\{1, 2, \ldots, m\}$, where $m'$ is the maximum number of documents in the outsourcing document set. The numbers are used to act as the identifiers of documents, and they are also used to negotiate agreements between the fog node and cloud node.

(xiii) $T$—a collection of numbers initialized to an empty set. In the process of construct $Q$, $L$ acts as an indicator and we will discuss this in Section 4.4.

(xiv) $\mathcal{R}$—ciphertexts of the top-$k$-related documents, which are returned to the data user by the cloud server.

4.2. Construction of Document Index Vectors Based on the Enhanced TF-IDF Model. In this article, we construct the document vectors based on both the keyword weight and position weight. The definitions of them are presented as follows.

(I) Keyword Weight. In the field of information retrieval, the TF-IDF model [39] is widely used to calculate the correlation between keywords and documents. TF-IDF model consists of two parts: keyword frequency and inverse document frequency. In order to express the correlation score between
a given keyword $w_j$ and a document $F_i$, in this article, we use the following formula to calculate it:

$$\text{Score}(w_j, F_i) = \frac{f_{i,w_j}}{|F_i|} \ln \left( \frac{|\mathcal{F}|}{|\mathcal{F}_w|} + 1 \right),$$

where $f_{i,w}$ represents the number of keywords $w_i$ in document $F_i$, $|F_i|$ is the total number of keywords in document $F_i$, $|\mathcal{F}|$ is the total number of document set $\mathcal{F}$, and $|\mathcal{F}_w|$ indicates the number of documents in document set $\mathcal{F}$ that contains the keyword $w_j$.

(2) Position Weight. Existing schemes equally treat the keywords in different positions in a document. Therefore, each keyword in a document has the same effect when retrieving the interested documents based on relevance score. In fact, the keywords play different roles when they appear in the title, abstract, and main body of the same document. Therefore, to reflect the affection of keyword positions on the important degrees, this article sets different position weights for the keywords in three different regions of a document, that is, the title, abstract, and main body.

The position weight of the title is set to $g_1$, the position weight of the abstract is set to $g_2$, and the position weight of the main body is set to $g_3$. Moreover, the weights $g_1$, $g_2$, and $g_3$ meet the following conditions:

$$\begin{align} & g_1 > g_2 > g_3, \\
& g_1, g_2, g_3 \in [0,1], \\
& g_1 + g_2 + g_3 = 1. \end{align}$$

The same keyword may appear in three positions of the document, so it is necessary to comprehensively measure the weight score of keywords in three positions. In this article, $\rho_1, \rho_2, \rho_3$ are used to indicate whether a keyword appears in the title, abstract, and text, respectively. If it appears, the corresponding score is 1; otherwise, the score is 0. The total weight score of the keyword in the whole document can be calculated by the following formula:

$$Z = g_1 \cdot \rho_1 + g_2 \cdot \rho_2 + g_3 \cdot \rho_3.$$  

As an example, if a keyword $w_j$ appears in the title and text of document $F_i$, the position weight score of keyword $w_j$ in document $F_i$ is $Z_{ij} = g_1 + g_3$.

Based on the keyword weight and position weight, the data owner generates a document index vector $D_i$ for each document $F_i$. For the entry $D_i[j]$ in the vector $D_i$, it is calculated as follows:

$$D_i[j] = Z_{ij} \cdot \text{Score}(w_j, F_i).$$

It can be observed from the equation that if the corresponding keyword $w_j$ appears in the document, $F_i$, $D_i[j]$ is equal to a positive number. Otherwise, we set $D_i[j] = 0$.

4.3. Query Extension Based on Word2Vec. In real life, the data users may want to query the documents based on synonyms or they may provide some improper keywords in the query process if they are not familiar enough to a specific field [9, 28]. Most existing schemes did not take this into consideration, and they do not support the synonyms-based query. Moreover, if the data users cannot provide a set of accurate keywords, they cannot get the proper search results.

In this article, we solve this problem by query extension based on the Word2Vec [1, 5] tool.

Word2Vec is a neural network-based tool for computing word vectors, and it was designed by Google in 2013. Word2Vec can be efficiently trained on millions of dictionaries and datasets, and get the word vectors of training results. Word vectors can be used to calculate the similarities between the words. The smaller the distance between two words, the higher the similarity between them. In this article, the data owner uses Word2Vec as the training tool and uses the Skip-gram model [40, 41] to train the data to obtain the word vector for each word.

In the query process, the data user first provides a set of keywords and outsources the keywords to the fog node. Then, the fog node selects other several similar keywords based on vectors generated by the Word2Vec model. In this article, the number of extending keywords is the same as the number of keywords provided by the data user, as shown in Figure 2. This is reasonable considering that the accuracy is the highest in this case and we will present the simulation results in Section 7.

4.4. The Framework of Privacy-Preserving Document Retrieval. In this section, we present the overall document retrieval framework by mainly employing the functionalities of the fog node and cloud server, which contains six parts: Setup, GenIndex, Enc, GenTrapdoor, Search, and Dec.

(1) Setup ($1^i$) $\rightarrow$ $\mathcal{K}$

In the initialization phase, let $G = <\alpha>$ be a multiplicative cyclic group of large prime order $p$, which is generated and exposed by the data owner. Then, he needs to generate the secret key set $\mathcal{K}$, including (1) one randomly generated $(m + m')$-bit vector $S$ and (2) two $(m + m' + 1) \times (m + m' + 1)$ invertible matrices $M_1$ and $M_2$. Moreover, the symmetric secret keys in $K$ are generated by the data owner to encrypt the files independently.

(2) GenIndex ($\mathcal{F}, \mathcal{K}$) $\rightarrow$ $\mathcal{I}$

The data owner first calculates the correlation score $\text{Score}(w_j, F_i)$ between keyword $w_j$ and document $F_i$ according to the TF-IDF method. Then, the data
owner uses the position weight scoring method to calculate the weight score $Z_{ij}$ of the three positions of in document $F_i$. At last, he generates a document index vector $D_i$ for every document $F_i$, as discussed in Section 4.2, where $D_i[j] = Z_{ij} \cdot \text{Score}(w_j, F_i)$ if the corresponding keyword $w_j$ appears in the document $F_i$, and otherwise, we set $D_i[j] = 0$.

Then, the data owner extends the $m$-dimensional document index vector $D_i$ to the $(m + m’ + 1)$-dimensional vector $\overline{D}_i = D_i \| V_i$, where $V_i (i = 1, 2, \ldots, n)$ is an $(m’ + 1)$-dimensional vector generated by Algorithm 1. In this algorithm, $L = \{1, 2, \ldots, m’\}$, $Z_p = \{0, 1, \ldots, p - 1\}$ and three elements are returned including $V_i (i = 1, 2, \ldots, n)$. It can be inferred from Algorithm 1 that each $V_i$ includes two numbers $a_i$ and $-a_i$ selected from $Z_p = \{0, 1, \ldots, p - 1\}$. Meanwhile, all the other entries are set as 0. The positions of $a_i$ in different $V_i$ are always different from each other, because once a position is selected, it will be deleted from $L$.

To protect the privacy of document vector $\overline{D}_i$, we need to encrypt them. Specifically, the data owner first splits $\overline{D}_i$ into two random vectors $(\overline{D}_i, \overline{D}_i')$ according to the secret vector $S$. For each $j$, if $S[j] = 0$, then $\overline{D}_i[j] = \overline{D}_i[j]/2 + r$ and $\overline{D}_i'[j] = \overline{D}_i[j]/2 - r$, where $r$ is a random number. On the contrary, if $S[j] = 1$, then we set $\overline{D}_i[j] = \overline{D}_i'[j] = \overline{D}_i[j]$. At last, the searchable encrypted index is calculated as $I_i = E_{\mathcal{S},\mathcal{F}} (\overline{D}_i) = [M_1^{\mathcal{F}} \overline{D}_i, M_2^{\mathcal{F}} \overline{D}_i']$ by using matrix multiplication with the matrix $M_1$ and $M_2$ in $\mathcal{S}_K$.

In this article, we employ the search table structure to organize these document index vectors and this can improve the search efficiency. The data owner first needs to randomly select a set of numbers $x_{A_1}, x_{A_2}, \ldots, x_{A_n} \in \{0, p - 1\}$ as the identifiers of keyword $w_1, w_2, \ldots, w_m$, respectively. After this, the data owner puts the document set $\{F_{k_1}, F_{k_2}, \ldots, F_{k_n}\} \subseteq \{F_1, F_2, \ldots, F_n\}$ containing the keyword $w_j (j = 1, 2, \ldots, m)$ together in a row of the search table. Meanwhile, the encrypted index set $\{I_{k_1}, I_{k_2}, \ldots, I_{k_n}\} \subseteq \{I_1, I_2, \ldots, I_n\}$ is generated in a similar way. Finally, an example of the encrypted search table can be created and outsourced to the cloud server as shown in Table 1.

(3) Enc ($\mathcal{F}, K$) $\rightarrow$ $\mathcal{E}$

After constructing the document index vectors, the documents can be encrypted and sent to the cloud server. In this article, the data owner uses a secure symmetric encryption algorithm (e.g., AES) to encrypt the plaintext files in the document collection $\mathcal{F}$ one by one with different keys and at last outsources them to the cloud server.

(4) GenTrapdoor ($Q^*, \mathcal{S}_K, \mathcal{T}_c$) $\rightarrow$ $T_{Q^*}$

In the preparation phase of this system, the data owner also needs to send $\mathcal{E}$ to the fog node. Note that, $x_{B_i}$ is a random number bound to $w_j (i = 1, 2, \ldots, m)$.

In the query phase, when the fog node receives a query request $Q^*$ from a data user, It first uses Word2Vec to extend the semantics of keywords in $Q^*$ to obtain $\overline{Q}$. According to $\mathcal{F}$ and $\overline{Q}$, the search vector $Q$ can be constructed by the fog node as shown in Figure 2. Specifically, if $w_i \in \overline{Q}$, $Q[i]$ is set to 1; otherwise, $Q[i]$ is set to 0. Based on the TF-IDF model, we can calculate the relevance score between the query and a document by calculating $\overline{Q^*} D_i$.

To hide the exact keywords in the documents, a negotiate process between the fog node and cloud server is designed. When $w_i \in \overline{Q}$, the fog node selects a random number $x_{B_i}$ associated with $w_i (i = 1, 2, \ldots, m)$. Then, he computes $Y_{B_i} = x_{B_i} mod p$ and $H_i = \overline{Y}_{A_i} mod p$, and sends $(\overline{Y}_{B_i}, H_i)$ to cloud server for $i = 1, 2, \ldots, m$.

To defend the forward and backward threat, the fog node also needs to extend the $m$-dimensional search vector $Q$ to the $(m + m’ + 1)$-dimensional vector $\overline{Q} = Q[V^*]$ as given by the following formula:

$$V^* = [V^*[j]] = \begin{cases} 1, & \text{if } j = T_i \cup \{m’ + 1\}, \\ 0, & \text{otherwise}. \end{cases}$$

Then, $\overline{Q}$ is split into two random vectors as $[\overline{Q}, \overline{Q}’]$ based on the secret bit string vector $S$. Different to the split process of $\overline{D}_i$, if the $i$th bit of $S$ is 0, we have $\overline{Q’}[i] = \overline{Q’’}[i] = \overline{Q}[i]$; on the contrary, if the $i$th bit of $S$ is 1, then we set $\overline{Q’}[i] = \overline{Q}[i]2 + r’$ and $\overline{Q’’}[i] = \overline{Q}[i]/2 - r’$, where $r’$ is a random number. After that, the encrypted search vector $T_{\overline{Q}}$ is

**Algorithm 1: Generation algorithm of $V_i (L, \mathcal{T})$.**
generated as \( T_\mathcal{Q} = \{M_1^{-1}\mathcal{Q}', M_2^{-1}\mathcal{Q}'\} \) and they are sent to the cloud server.

5. Dynamic Update of Document Set

After the insertion or deletion of a document, we need to synchronously update the encrypted index table stored in the cloud server. The dynamic update process of our scheme comprises three phases, that is, generation of update information, update operations on the cloud server, and update operations on the fog node. We design three algorithms corresponding to the three phases, and they are discussed in the following:

(i) \( \{\text{Stable}', C_i, T\} \leftarrow \text{GenUpdateInfo.} \quad (K, \text{Stable}, X_F, L, \text{updtype}) \)

This algorithm generates the update information \( \{\text{Stable}', C_i\} \) and \( T \), which will be sent to the cloud server and fog node, respectively. In order to reduce the communication overhead, we assume that the data owner stores a copy of unencrypted index table for all the documents. Here, we employ the updtype \( \in \{\text{Ins, De } l\} \) to denote as either an insertion operation or a deletion operation for the document \( F_i \). We employ the Stable' to denote as the updated table. The specific process is presented as follows. The notion \( X_F \) represents the set of keyword identifiers included in document \( F_i \).

(ii) If updtype is equal to \( \text{De } l \), the data owner deletes the document \( F_i \) from the document collection and finds out all the keyword identifiers in the document \( F_i \), and they form a set, \( X_F \), of keyword identifiers. Then, the data owner deletes the encrypted index \( I_i \) corresponding to the keyword identifier \( X_F \) in the encrypted index table Stable and generates a new encrypted index table Stable'. The data owner uploads \( \{\text{Stable}', C_i\} \) to the cloud server. Meanwhile, the data owner adds \( v_i \) to the set \( T \), which is sent to the fog node.

(iii) If updtype is equal to \( \text{Ins} \), the data owner first adds the new document \( F_i \) to the document set and finds out all the keyword identifiers set \( X_F \) in the document \( F_i \). Then, the data owner adds the encrypted index \( I_i \) corresponding to the keyword identifier \( X_F \) in the encrypted index table Stable and generates a new encrypted index table Stable'. The data owner encrypts the document \( F_i \) with secret key \( K \) to obtain \( C_i \) and uploads \( \{\text{Stable}', C_i\} \) to the cloud server. Subsequently, the data owner generates a vector \( V_i \) for the document \( F_i \) and deletes the element \( v_i \) that determines the position of \( a_i \) in \( V_i \) from the set \( T \). Finally, the data owner sends \( L \) to the fog node.

(2) \( \{\text{Stable}', \mathcal{C}'\} \leftarrow \text{Update.} \quad (\text{Stable}, \mathcal{C}, \text{updtype}, c_i) \)

In this algorithm, the cloud server updates the search table and encrypted document set based on the received information from the data owner. Specifically, the cloud server always first replaces the outdated search table Stable with the new search table Stable'. This process is irrelevant to the update type of De \( l \) or Ins.
In the process of updating the encrypted document set, the cloud server needs to check the type of update first. If updtype is equal to Ins, cloud server inserts the encrypted document \( C_i \) into \( \mathcal{E} \), obtaining a new collection \( \mathcal{E}' \). If updtype is equal to De I, cloud server deletes the encrypted document \( c_i \) from \( \mathcal{E} \) to get a new collection \( \mathcal{E}' \).

\[ (3) \qquad \{ \mathcal{T}' \} \leftarrow \text{Update.} \quad \text{(updtype,} \mathcal{T} \text{)} \]

The update process in the fog node is quite straightforward. In this algorithm, fog node replaces \( \mathcal{T} \) by \( \mathcal{T}' \). Then, in the future, the updated \( \mathcal{T}' \) is employed to generate new trapdoors. In this way, our scheme can defend the forward and backward attack.

Through the above process of deleting or inserting documents, it shows that the dynamic update process of our scheme is easy and hence the scheme is of great scalability. And we will analyze the security of the scheme in the next section.

### 6. Security Analysis

As discussed in Section 3.3, we mainly consider the security of documents, privacy of document vectors, and trapdoors. Moreover, we also consider the forward and backward security of our scheme. They will be presented in the following.

#### 6.1. Document Privacy

In this article, the mature symmetric encryption algorithm AES is used to encrypt the document set \( \mathcal{F} \), and the ciphertext document set \( \mathcal{E} \) is generated and uploaded to the cloud server. Note that, the secret keys for different documents are different and independent with each other in the encryption process. Even the cloud server can successfully get a set of the secret keys, they cannot infer the other keys based on the ciphertexts. Therefore, the proposed scheme can effectively ensure the security of the document content and defend against the known ciphertext threat model.

#### 6.2. Privacy of Document Vectors and Trapdoors

In terms of document vectors and trapdoors, the data owner needs to randomly generates a bit string \( S \) and two reversible matrices \( M_1 \) and \( M_2 \). After the extension, the extension index vector \( \mathcal{D} \) and the extension search vector \( \mathcal{Q} \) are first split based on \( S \). Then, we generate the encrypted index \( I_i \) and the search request trapdoor \( T_{\mathcal{Q}} \) based on \( M_1 \) and \( M_2 \). At last, the encrypted vectors rather than plaintext vectors are uploaded to the cloud server. Because the space of the key matrix is very large, the probability that the untrusted cloud server can correctly forge the key matrix to crack the encrypted index and the search request trapdoor is almost zero. An efficient manner is constructing equation groups based on the background information and accurately calculates the two matrices. However, as proved in Ref. [14], the adversary cannot construct enough equations to calculate all the unknown variables in the matrices. Therefore, the security of information contained in document index and search vector is effectively guaranteed. In conclusion, since the untrusted cloud server can only obtain ciphertext document set, encrypted index, and search request trapdoor, it cannot obtain any useful plaintext information. Therefore, our scheme is safe for the known ciphertext model.

In order to further prevent the untrusted cloud server from mining and disclosing the document privacy information according to the known background knowledge, that is, according to the internal relationship between document index and retrieval trapdoor, we also mislead the adversary by dynamically changing the appearance of the trapdoors. Our article uses the split indicator \( S \) in the key \( \mathcal{K} \). In this article, the extension index vector \( \mathcal{D} \) and the extension search vector \( \mathcal{Q} \) are randomly split by using the split indicator \( S \) in key \( \mathcal{K} \), and the random numbers \( r \) and \( r' \) are introduced to ensure that there is no correlation between multiple document indexes and search vectors. In other words, even if the user employs the same query request multiple times, the search request trapdoor received by the untrusted cloud server is different. This effectively resists the attack of statistical analysis. Therefore, our proposed scheme is also safe for the known background knowledge threat model.

#### 6.3. Forward and Backward Security

As discussed in Section 1, forward and backward security is of great importance although most existing encrypted document retrieval schemes did not consider this threat. Our scheme can properly defend this attack and we prove this in the following.

In real life, the cloud server is not trusted, and it may save the trapdoors that have been searched or documents that have been deleted. If so, the untrusted cloud server can use the outdated query vectors to retrieve the newly inserted documents and use the new query to retrieve the deleted documents. In our scheme, when the query request is valid, the extended part of the vector \( V_i \cdot V' = 0 \); then, \( T_{\mathcal{Q}} \cdot I_i = Q \cdot D_i \); however, when the query request is illegal, the extended part of the vector \( V_i \cdot V' = -a \); then, \( T_{\mathcal{Q}} \cdot I_i = Q \cdot D_i - a_i \). As can be seen from the above equation, when the query is illegal, the correct correlation score will be confused by the randomness factor \( a_i \).

Without the loss of generality, we consider a simple document collection with 2 files \( \{F_1, F_2\} \). We assume that the maximal number of outsourced documents is 4, that is, \( m' = 4 \). In the initial phase, we set \( L = \{1, 2, \ldots, m'\} = \{1, 2, 3, 4\} \) and \( \mathcal{T} = \emptyset \).

The data owner needs to extend the document vectors of \( D_1 \) and \( D_2 \) based on Algorithm 1 as follows:

\[
\begin{align*}
\mathcal{D}_1 &= D_1 || V_1, \\
\mathcal{D}_2 &= D_2 || V_2,
\end{align*}
\]

where we assume that \( V_1 = (0, a_1, 0, 0, -a_1) \) and \( V_2 = (0, 0, 0, a_2, -a_2) \). Based on the definition of \( L \) and \( \mathcal{T} \), we can calculate them as follows:

\[
\begin{align*}
L &= \{1, 2, 3, 4\} - \{v_1, v_2\} = \{1, 2, 3, 4\} - \{2, 4\} = \{1, 3\}, \\
\mathcal{T} &= \emptyset + \{v_1, v_2\} = \{2, 4\}.
\end{align*}
\]
Note that, $\mathcal{L} = \{2, 4\}$ is delivered to the fog node. At last, the extended document vectors $\mathcal{D}_1$ and $\mathcal{D}_2$ are split and encrypted as follows:

\[
\begin{align*}
I_1 &= E_{SK}(\mathcal{D}_1) = \{M^T\mathcal{D}_1, M_2^T\mathcal{D}_1\}, \\
I_2 &= E_{SK}(\mathcal{D}_2) = \{M_1^T\mathcal{D}_2, M_2^T\mathcal{D}_2\}.
\end{align*}
\]

The encrypted index vectors $I_1$ and $I_2$ are sent to the cloud server.

In a query process, the fog node also needs to extend the search vector $Q$ based on $\mathcal{L}$ as follows:

\[
\mathcal{Q} = Q \cup V',
\]

where $V' = (0, 1, 0, 1, 1)$. This is reasonable considering that $\mathcal{L} = \{2, 4\}$ and we can infer that $V'[2] = V'[4] = V'[5] = 1$. Then, the extended search vector $\mathcal{Q}$ is split and encrypted as follows:

\[
T_{\mathcal{Q}} = \{M_1^{-1}\mathcal{Q}, M_2^{-1}\mathcal{Q}''\}.
\]

At last, $T_{\mathcal{Q}}$ is sent to the cloud server.

Now, we consider the dynamic of the document collection and analyze the forward and backward security of our scheme. When the data owner inserts a new document $F_3$ into the document collection, its vector $D_3$ needs to be extended as follows:

\[
\mathcal{D}_3 = D_3 \cup V_3,
\]

where we assume that $V_3 = (a_2, 0, 0, 0, -a_3)$. Meanwhile, we can calculate $L = \{1, 3\} - \{V_3\} = \{1, 3\} - \{1\} = \{3\}$ and $\mathcal{L} = \{2, 4\} + \{V_2\} = \{1, 2, 4\}$ by Algorithm 1. Then, the data owner sends $\mathcal{L} = \{1, 2, 4\}$ to the fog node. Then, the extended document vector $\mathcal{D}_3$ is split and encrypted by the data owner as follows:

\[
I_3 = E_{SK}(\mathcal{D}_3) = \{M_1^T\mathcal{D}_3, M_2^T\mathcal{D}_3\}.
\]

We further assume that the data owner deletes a document $F_2$, and then, it needs to update $L$ and $\mathcal{Q}$ as follows:

\[
\begin{align*}
L &= \{L\} + \{v_2\} = \{3\} + \{4\} = \{3, 4\} \\
\mathcal{L} &= \{1, 2, 4\} - \{v_2\} = \{1, 2\}.
\end{align*}
\]

Subsequently, the data owner sends the updated $\mathcal{L}$ to the fog node.

For a new search vector $Q$, the fog node extends it based on $L = \{1, 2\}$ as presented in the following:

\[
\mathcal{Q} = Q \cup V',
\]

where $V' = (1, 1, 0, 0, 1)$. After, splitting and encryption, the fog node sends $T_{\mathcal{Q}}$ to the cloud server.

The dishonest cloud server may not actually delete the document $F_2$, and it wants to calculate the relevance between the new trapdoor and $F_2$. However, the relevance score cannot be accurately calculated, because $I_3 \cdot T_{\mathcal{Q}} = Q \cdot D_3 - a_3$, and there is always a nonnegligible error.

It can be observed from the above process that our scheme can defend against the forward and backward security and properly protect the privacy information of the documents. Therefore, the scheme well meets the design goals we put forward.

### 7. Performance Evaluation

In this section, we first analyze the time complexity of the designed SES scheme, which shows the efficient of the scheme. Secondly, we implement the proposed scheme using python language on a computer with 2.9 GHz Intel Core, Windows 10 operation system, and 16G memory. We compare the performance of the proposed scheme with that of the MRSE scheme, PRSE scheme, and MRSE-HCI scheme. In experiments, we randomly select 5000 files from the Request for Comments Dataset (RFC) [42] and select in total 2839 keywords from the document files. In the following, we evaluate the performance of our scheme in terms of the index construction efficiency, document retrieval accuracy, and document retrieval efficiency. Each simulation is repeated 10 times, and the average simulation results are presented and analyzed.

#### 7.1. Time Complexity Analysis

Firstly, as shown in Table 2, we analyze the time complexity of the SES scheme proposed in this article and the MRSE_HII_TF [7] scheme proposed by Cao et al. Both schemes adopt a secure KNN architecture and TF-IDF idea. Among them, $L_{ES}$ represents the average length of documents. And $T_{AES}, T_{W2Y}, T_{M}$ represent the

<table>
<thead>
<tr>
<th>Identifier</th>
<th>The searchable encrypted index</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_{A_1}$</td>
<td>$I_{A_1}$</td>
</tr>
<tr>
<td>$x_{A_2}$</td>
<td>$I_{A_2}$</td>
</tr>
<tr>
<td>$x_{A_n}$</td>
<td>$I_{A_n}$</td>
</tr>
</tbody>
</table>

### Table 1: An example of the search table.

We further assume that the data owner deletes a document $F_2$, and then, it needs to update $L$ and $\mathcal{Q}$ as follows:

\[
\begin{align*}
L &= \{3\} + \{v_2\} = \{3\} + \{4\} = \{3, 4\} \\
\mathcal{L} &= \{1, 2, 4\} - \{v_2\} = \{1, 2\}.
\end{align*}
\]

Subsequently, the data owner sends the updated $\mathcal{L}$ to the fog node.

For a new search vector $Q$, the fog node extends it based on $L = \{1, 2\}$ as presented in the following:

\[
\mathcal{Q} = Q \cup V',
\]

where $V' = (1, 1, 0, 0, 1)$. After, splitting and encryption, the fog node sends $T_{\mathcal{Q}}$ to the cloud server.

The dishonest cloud server may not actually delete the document $F_2$, and it wants to calculate the relevance between the new trapdoor and $F_2$. However, the relevance score cannot be accurately calculated, because $I_3 \cdot T_{\mathcal{Q}} = Q \cdot D_3 - a_3$, and there is always a nonnegligible error.

It can be observed from the above process that our scheme can defend against the forward and backward security and properly protect the privacy information of the documents. Therefore, the scheme well meets the design goals we put forward.

#### 7.1. Time Complexity Analysis

Firstly, as shown in Table 2, we analyze the time complexity of the SES scheme proposed in this article and the MRSE_HII_TF [7] scheme proposed by Cao et al. Both schemes adopt a secure KNN architecture and TF-IDF idea. Among them, $L_{ES}$ represents the average length of documents. And $T_{AES}, T_{W2Y}, T_{M}$ represent the
time consumption for AES algorithm encryption and decryption, Word2Vec expansion algorithm, and power operation in multiplicity group, respectively. The time complexity analysis of the algorithm is as follows.

In the algorithms Setup, GenIndex, Enc, and Dec, the proposed SES scheme has similar time complexity to the MRSE scheme. It is worth noting that the $U$ in the MRSE scheme is a parameter that can be set at will. When $U = n$, the time complexity of the two schemes will be exactly the same.

In the GenTrapdoor algorithm, $\Theta(m^2 + n^2 + 2mn)$ and $\Theta(m^2 + U^2 + 2mnU)$ are roughly similar (they are exactly the same when $U = n$). Therefore, it can be considered that in the GenIndex step, the time complexity of the SES scheme is $\Theta(T_{W2V}) + \Theta(2mT_M)$ more than that of MRSE. However, considering that for data users, GenIndex and search steps are performed by fog nodes and clouds, respectively, we should pay attention to the time interval between users sending retrieval requests and receiving retrieval results, that is, the time taken by the sum of the two steps. In the algorithm Search, it is noted that each query contains few keywords and the number of relevant documents $k$ is much less than $n$. This means that $\Theta(km + kn)$ at least consumes $\Theta(0.5mn + 0.5nr^2)$ less time than $\Theta(mn + nr^2)$. Moreover, $\Theta(0.5mn + 0.5nr^2) > \Theta(T_{W2V}) + \Theta(2mT_M) + \Theta(mT_M)$. Therefore, when we consider algorithms GenTrapdoor and Search together, the total time complexity is $\Theta(T_{W2V}) + \Theta(2mT_M) + \Theta(m^2 + n^2 + 2mn) + \Theta(mT_M) + \Theta(kl) < \Theta(m^2 + U^2 + 2mnU) + \Theta(nm + n^2) + \Theta(nlogn)$, which means that the efficiency of retrieval using the SES scheme is higher than that of MRSE.

To sum up, through the time complexity analysis of the SES scheme and MRSE scheme, it can be clarified that our scheme has higher efficiency and shorter retrieval waiting time, which is due to our use of the Diffie–Hellman key agreement algorithm to optimize the retrieval algorithm. Next, we will use simulation experiments to verify our analysis.

### 7.2. Trapdoor Construction Efficiency

In our scheme, the keywords in query requests are first extended based on the Word2Vec model. According to the analysis in Section 7.1, the time consumption of constructing the encrypted query vectors in our scheme is slightly enlarged by the extension process. As shown in Figure 3, the total time consumption comprises two parts, that is, the time of the extension process and that of the encryption process.

It can be observed from Figure 3 that with the increase of the number of extended keywords, the time consumption monotonously increases. This is reasonable considering that some extra time is consumed to extend the keyword set; meanwhile, the time consumption of encryption keeps stable. This can be explained by the fact that the length of the vectors is approximately constant. Compared with the total time consumption, the extra time consumption is very slight and it can be ignored. In the extreme case, that is, the keywords are extended by five times, the time consumption is only 3.989 ms and it is about 8% of the total time consumption.

### 7.3. Document Retrieval Accuracy

In this article, we measure the document retrieval accuracy by the proportion of
related documents in all the returned results. In each query, the top-20 related documents are returned and search accuracy with different extension parameters is presented in Figure 4. With the increase of the extension parameter, the search accuracy increases in the initial phase and then decreases. When we set the extension parameter as 2, the search accuracy is the highest and it is about 91.5% as shown in Figure 4. This is reasonable considering that when the number of extended keywords is small, the extended keywords can provide extra information for document search. However, if the number of the extended keywords is too large, some random noises are also added to the search process and they may mislead the search results. As a consequence, the search accuracy decreases. In the following experiments, we extend the keywords by 2 times.

We further compare the search accuracy of our scheme to that of MRSE. It can be observed from Figure 5 that the search accuracy of our scheme is always higher than 90% with the increase in the number of documents. However, the search efficiency of MRSE gradually decreases from nearly 90% to 80%. In conclusion, the employment of keywords extension process greatly improves the search accuracy in the encrypted cloud file systems.

7.4. Document Retrieval Efficiency. In this section, we compare our scheme with MRSE, MRSE-HCI, and PRSE in terms of document retrieval efficiency. As shown in Figure 6, with the increase in the number of documents in the collection, the search time of all the four schemes increases. The search time of MRSE increases in an approximately linear manner. This is reasonable considering that all the document vectors need to be scanned to find the search results. The schemes of MRSE-HCI and PRSE perform better, and however, the performance of the proposed scheme is the best. This can be explained by the fact that a novel index
structure, that is, the search table, is employed to improve the retrieval efficiency.

Except for the number of documents, the number of keywords also affects the average search time. We further compare the total time consumption of trapdoor construction plus document retrieval of our scheme with that of MRSE scheme, PRSE scheme and MRSE-HCI scheme. The number of total keywords is set as 2839, and the keywords are extended by 2 times in which case the search accuracy is the highest. As shown in Figure 7, the waiting time for the data user retrieval of MRSE, PRSE, MRSE-HCI, and our scheme increases with the increase in the number of keywords in the query. However, it is worth noting that our scheme and MRSE-HCI have a lower retrieval delay among the four schemes. Especially when the number of keywords in the query is less than 20, our scheme has the best document retrieval efficiency, because it is impossible for users to enter so many keywords for retrieval.

8. Conclusion and Future Work

In this article, we propose a secure and efficient encrypted document retrieval scheme based on both fog and cloud computing. A fog node is employed to act as a bridge between the cloud server and data users. The workloads of data owner and data users decrease, and meanwhile, the level of system security improves. We modify the TF-IDF model by taking the position information of keywords into consideration, and it can reflect the importance degrees of the keywords in a more accurate manner. In this way, the document vectors are more reasonable. To always return the proper search results even for imperfect query requests, the keywords in query requests are first extended based on Word2Vec before constructing query vectors. We employ the secure kNN algorithm to encrypt these document vectors and query vectors. Therefore, the security of the vectors greatly increases while maintaining their searchability. Forward and backward security is also considered and properly protected in our scheme. An extra vector is appended after the document vector to prevent the cloud server from querying the deleted documents by a new trapdoor or querying the new documents by an outdated trapdoor. At last, an index structure is designed to improve search efficiency. The security of our scheme is carefully analyzed from several aspects. Analysis and simulation results illustrate that our scheme can provide secure and efficient document retrieval service over encrypted cloud file systems to data users.

In our future work, we would like to apply neural network architecture in the retrieval process, which may make the rank order more accurate. Also, how to improve the retrieval efficiency is a subject for future work.

Data Availability

The figures and tables used to support the findings of this study are included in the article.

Conflicts of Interest

The authors declare no conflicts of interest.

Acknowledgments

This work was supported by the National Natural Science Foundation of China under Grant numbers: 62001055 and 62102017, and the Fundamental Research Funds for the Central Universities (YWF-22-L-1273).

References


[38] S. N. Shirazi, A. Gougildis, D. Hutchinson, and D. Hutchison, "The extended cloud: review and analysis of mobile edge computing and fog from a security and resilience perspective,"


