symmetry

MDPI

Article

Fast, Searchable, Symmetric Encryption Scheme Supporting Ranked Search

Wei He [1], Yu Zhang [1,*] and Yin Li [2]

[1] School of Computer and Information Technology, Xinyang Normal University, Xinyang 464000, China; weihe@xynu.edu.cn
[2] School of Cyberspace Security, Dongguan University of Technology, Dongguan 523820, China; liyin@dgut.edu.cn
* Correspondence: willow1223@126.com or zhangyu86@xynu.edu.cn; Tel.: +86-136-6729-0954

Abstract: Searchable encryption (SE) is one of the effective techniques for searching encrypted data without decrypting it. This technique can provide a secure indexing mechanism for encrypted data and utilize a secure trapdoor to search for the encrypted data directly, thus realizing a secure ciphertext retrieval function. Existing schemes usually build a secure index directly on the whole dataset and implement the retrieval of encrypted data by implementing a secure search algorithm on the index. However, this approach requires testing many non-relevant documents, which diminishes the query efficiency. In this paper, we adopt a clustering method to preclassify the dataset, which can filter out quite a portion of irrelevant documents, thus improving the query. Concretely, we first partition the dataset into multiple document clusters using the k-means clustering algorithm; then, we design index building and searching algorithms for these document clusters; finally, by using the asymmetric scalar-product-preserving encryption (ASPE) scheme to encrypt the indexes and queries, we propose a fast searchable symmetric encryption scheme that supports ranked search. Detailed security analysis demonstrates that the proposed scheme can guarantee the data and query security of the search process. In addition, theoretical and experimental analysis indicates that our scheme outperforms other similar schemes in terms of query efficiency.

Keywords: searchable symmetric encryption; searchable encryption; keyword search; ranked search; search over encrypted data

Citation: He, W.; Zhang, Y.; Li, Y. Fast, Searchable, Symmetric Encryption Scheme Supporting Ranked Search. *Symmetry* **2022**, *14*, 1029. https://doi.org/10.3390/sym14051029

Academic Editor: Yu-Chi Chen

Received: 19 April 2022
Accepted: 15 May 2022
Published: 18 May 2022

1. Introduction

With the rapid growth of cloud computing, more and more people and businesses are willing to outsource their data to the cloud. Using cloud computing, cloud service providers can provide online data storage and analysis services to different types of customers, which greatly reduces the cost of data storage and computing for users. As a result, data outsourcing techniques in the cloud environment have become widely popular. However, although data outsourcing brings convenience to data users, it also inevitably faces the risk of data leakage. To ensure data security, we can encrypt the data before uploading. However, encryption makes ciphertext no longer support retrieval operations as plaintext does. Driven by the demand for searching over encrypted data, a large number of searchable encryption schemes emerged [1–6]. According to different encryption mechanisms, searchable encryption schemes can be divided into two categories: searchable symmetric encryption (SSE) [1–3] and searchable public key encryption (SPE) [4–6]. SPE scheme can provide secure search service for multiple data owners, but public key operation has high computing cost and is not suitable for large-scale data application scenarios. On the contrary, symmetric encryption in the SSE scheme has simple operation and low cost, which is more suitable for the current big data environment.

The SSE scheme can solve the problem of finding documents most related to the query keywords in the corpus according to a given metric mechanism. Earlier SSE schemes only

return documents containing all the query keywords and did not return documents based on how closely they relate to the query keywords. To improve query accuracy, Cao et al. proposed an SSE scheme supporting ranked search using the term frequency-inverse document frequency (TF-IDF) model [7]. However, due to the use of the forward index structure, the search time is linear with the number of documents, which is not practical in the big-data environment. To improve the search efficiency, by using a binary balanced tree structure, Xia et al. proposed an SSE scheme with a sublinear search complexity [8]. Subsequently, Guo et al. used bloom filter technology to compress the vector dimension of internal tree nodes [9], further improving the query efficiency of the SSE scheme.

Although these two schemes have achieved ranked search on encrypted data, there is still a room for improvement. As the number of documents increases, the number of nodes in the index tree increases dramatically, resulting in high time cost to search the index tree. In order to improve the query efficiency, in this paper, we propose a more efficient SSE scheme compared with previous schemes with similar functions. To implement our scheme, we first transform the documents into the corresponding semantic vectors using a keyword conversion method. Then, the document set is divided into multiple document clusters using the k-means clustering algorithm with document semantic vectors, and index trees are built for each document cluster. Finally, we exploit the ASPE scheme to encrypt these index trees and user queries to achieve a secure ranked search. To sum up, the contribution of this paper comprises the following parts.

(1) A clustering algorithm is adopted to cluster the document set into multiple document clusters, and a secure index tree is constructed on each cluster. By utilizing this approach, the time complexity of index tree retrieval can be reduced because the height of the index tree is decreased.
(2) We optimize the search method to reduce the number of index trees to be retrieved and further improve the query efficiency of our scheme without sacrificing query accuracy too much.
(3) By utilizing the ASPE scheme [10] to encrypt the index and the query, we propose a fast SSE scheme support ranked search (F-SSE-RS). Moreover, we also design a dynamic update method so that the index in our scheme can support safe document update operations.

In addition, we give a detailed analysis to prove the security of F-SSE-RS and implement it on a widely used data collection. The experiment results demonstrate that our scheme is feasible in practice.

Related Work: Searchable encryption, as an attractive technique in data security and privacy protection, has received much attention in recent years. According to its different cryptographic prototypes, there are two main categories: searchable symmetric key encryption (SSE) and searchable public key encryption (SPE).

In the SSE scheme, the data owner and the data user share the secret key. Song et al. proposed the first SSE scheme that supports only a single keyword search [1]. To realize multi-keywords search, Goh designed an SSE scheme that supports conjunctive keyword search using a technique called the bloom filter [11]. Subsequently, to support more flexible query conditions, researchers proposed some SSE schemes that support complex query conditions such as range search [12,13], fuzzy search [14,15], and semantical search [2,16]. However, since these schemes fail to measure the relevance of documents to queries, many low-relevance documents will be returned, thus resulting in a large computational and communication overhead. To solve this problem, Wang et al. proposed an SSE scheme that supports ranked search in [17]. This scheme can compute the relevance between documents and queries in the encrypted state and sort the query results. Since the scheme proposed in [17] only supports a single keyword search, Cao et al. proposed a ranked search scheme that supports multiple keywords search [7]. However, this scheme adopts a forward index structure in which each document has an individual index, so its search time is linear in the number of documents. The search efficiency of this scheme is impractical in the current big data scenario. To improve the search efficiency, by using a tree–index structure, Xia et al.

proposed a similar SSE scheme [8]. The search efficiency of this scheme is sublinear with the number of documents. Subsequently, Guo et al. further improved the scheme to reduce the index building time [9]. However, we find that there is still room for decreasing the search time of these schemes, so we will propose a new scheme to improve the query efficiency in this paper. Besides, many recent SSE schemes are dedicated to supporting more complex query conditions, such as fuzzy query [18], spatial data query [19], and phrase query [20]. These schemes can better meet the user's query intent.

In the SPE scheme, the data owner uses the public key to build an encrypted index, while the data user uses the private key to perform ciphertext retrieval. Thus, SPE naturally supports many-to-one query scenarios. Boneh et al. introduced the first SPE scheme, called public key encryption with keyword search (PEKS) [4]. However, it supports only single keyword retrieval. Subsequently, researchers have proposed various SPE schemes that support multiple keywords queries, such as conjunctive keyword search [21], disjunctive keyword search [22], and Boolean keyword search [23]. In recent years, in addition to the research on multi-keyword search, many studies have targeted the security, precision, and efficiency of SPE. For example, access control [24], fast query [25], and semantic search [26]. All these works have greatly improved the usefulness of SPE.

Organization: The rest of this paper is organized as follows. In Section 2, we define the system model and the threat model of the scheme and give the design objectives of our scheme. Section 3 focuses on the index building and search method of the proposed scheme. Section 4 gives the construction process of the F-SSE-RS scheme and also gives the updated algorithm of the scheme as well as the security analysis. The theoretical and experimental analysis of the proposed scheme is presented in Section 5. The conclusion is given in Section 6.

2. Problem Formulation

This section first presents the system model of the F-SSE-RS scheme. Then, based on this model, we introduce two threat models commonly considered in SSE schemes. Finally, we propose the design goal of our scheme. For clarity, the main notations utilized in this paper are listed in Table 1.

Table 1. Description of the main notations in the F-SSE-RS scheme.

F	A document set $\{f_1, f_2, \ldots, f_d\}$.
d	The number of documents in F.
$DIC = \{w_1, w_2, \ldots, w_N\}$	The dictionary of a dataset.
N	The number of keywords in the dictionary.
$W_i = \{w_{i1}, w_{i2}, \ldots, w_{in_i}\}$	The keyword set for the document, f_i, where $i \in [1, d]$.
n_i	The number of keywords in W_i, where $i \in [1, d]$.
w_{ij}	The j-th keyword in W_i, where $i \in [1, d]$, $j \in [1, n_i]$.
$\overrightarrow{W_i}$	The vector representation for W_i.
$Q = \{q_1, q_2, \ldots, q_m\}$	A keyword query.
q_i	A keyword in Q, where $i \in [1, m]$.
$\overrightarrow{Q}$	The vector representation of the query Q.
T_Q	The trapdoor of Q.
$C = \{c_1, c_2, \ldots, c_k\}$	k document clusters divided from F.
$c_i = \{f_{i1}, f_{i2}, \ldots, f_{i\phi}\}$	The document set in c_i.
f_{ij}	The j-th document in the cluster, c_i, where $i \in [1, k]$, $j \in [1, \phi]$.
$\overrightarrow{W_{ij}}$	The vector representation of f_{ij}.

Table 1. *Cont.*

k	The number of clusters for dataset clustering.
ϕ	The number of documents in each cluster.
T_i	An index tree for the cluster, c_i, where $i \in [1,k]$.
r_i	The root node for T_i, where $i \in [1,k]$.
u	A node in an index tree.
$\vec{u}$	The vector representation of the node u.
$Ind = \{r_1, r_2, \ldots, r_k\}$	The index for F.
$E_{Ind} = \{E_{T_1}, E_{T_2}, \ldots, E_{T_k}\}$	The encrypted index for F.
E_{T_i}	The encrypted index tree for the cluster, c_i, where $i \in [1,k]$.
t	The number of index trees needed to be search.
θ	The number of documents needed to be returned.

2.1. System Model

Data owner (DO), data user (DU), and cloud server (CS) are three different roles in the system model of F-SSE-RS. To further illustrate the relationships among these three roles, we present an architectural diagram of the system model in Figure 1. As shown in Figure 1, DO encrypts a collection of documents and builds a safe index before sending them to DU. Then, DO sends these encrypted documents to CS together with the encrypted index. Once DU wishes to run a query, Q, it computes and transmits a search trapdoor, T_Q, of Q to CS. After getting the trapdoor, CS checks the encrypted index using T_Q and sends the most relevant documents to DU.

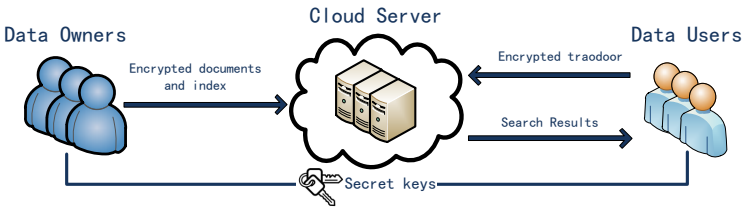

Figure 1. System model of F-SSE-RS.

To clarify the system model, the specific duties for each role are formally described as follows.

(1) Data owner (DO): DO owns a large number of sensitive documents $F = \{f_1, f_2, \ldots, f_d\}$. DO utilizes a symmetric key encryption scheme, e.g., AES, to encrypt the document set F, and adopts the F-SSE-RS scheme to build the secure searchable index. After these operations, DO uploads encrypted documents and secure index to CS. Finally, DO delivers the secret key to the data users who have been granted access to the data.

(2) Data user (DU): An authorized DU can make a secure query over encrypted data. Given a query Q, DU creates a trapdoor with the secret key and Q and sends it to CS. When DU gets search results from CS, DU can use the secret key to decode the encrypted contents.

(3) Cloud server (CS): DO's encrypted index and documents are stored in CS. Once the trapdoor, T_Q, is obtained from DU, CS executes the query over the index and returns the most relevant encrypted documents related to Q. In addition, CS runs update operations over the encrypted index after obtaining updated information from DO.

2.2. Threat Model

In reality, most cloud servers are not completely trustworthy, which means they might be snooping on information they should not have access to. Here, we suppose that the

cloud server is an "honest-but-curious" model, adopted by many SE schemes [7–9]. In the honest-but-curious model, CS performs algorithms of the F-SSE-RS scheme in terms of the desired computational process, but CS infers the user's privacy information curiously. Under the honest-but-curious model, based on the extent of information known to CS, our scheme adopts the following two threat models proposed by Cao et al. [7].

- **Known ciphertext model:** In this model, CS only knows encrypted documents and the secure index, which are stored on the server.
- **Known background model:** CS can access more information in this model than the aforementioned model. This information involves the relationship between a trapdoor and the dataset, and the statistics related to the dataset. For example, CS might exploit the dataset's term frequency (TF)-inverse document frequency (IDF) knowledge to perform the statistical attack.

2.3. Design Goals

To make the ranked search execute efficiently and securely, our scheme should concurrently satisfy the following goals under the aforementioned model.

(1) **Multi-keyword ranked search:** Each document, f_i, in F is associated with a keyword set, $W_i = \{w_{i1}, w_{i2}, \ldots, w_{in_i}\}$, in which n_i is the number of keywords in W_i. The multi-keyword query, Q, is $Q = \{q_1, q_2, \ldots, q_m\}$. The F-SSE-RS scheme's search result is sorted, which means that F-SSE-RS returns documents whose keyword set, W_i, is highly relevant to the query, Q. Furthermore, the F-SSE-RS scheme can enable efficiently dynamic activities, such as document insertion and deletion.

(2) **Efficiency:** The F-SSE-RS scheme can achieve sublinear search efficiency. Furthermore, the time cost of keyword search is substantially lower than existing similar schemes.

(3) **Privacy preserving:** The F-SSE-RS scheme, like some previous schemes, prevents CS from deducing more private information from ciphertexts, secure indexes, and trapdoors. Privacy requirements that our scheme focuses on are listed as follows.

   - Document and index privacy: Document privacy is usually protected by traditional symmetric-key encryption schemes, e.g., AES, DES, and six-face cubical key encryption [27]. For index privacy, the F-SSE-RS scheme prevents CS from learning what is hidden in the index.
   - Trapdoor unlinkability: The trapdoor generation algorithm needs to be probabilistic rather than deterministic, which means that the same keyword query will generate different trapdoors.
   - Keyword privacy: Although the trapdoor can be protected using cryptographic techniques, search results can be adopted to infer query keywords. Thus, our scheme needs to prevent CS from learning query keywords from trapdoors by search results and statistics of documents.

3. Methods for Index Building and Searching

In this section, we give the method for creating the plaintext index as well as the search method over this index. Before presenting these methods, we first devise a keyword conversion method to transform documents and queries into vector representations. Then, we introduce the index building method, which mainly consists of two steps: splitting the corpus into several document sets and building an index tree for each document set. After these two steps, the index tree of each document set is combined to construct the index of the entire corpus. Finally, we give a recursive method for searching the index. The following subsections provide a concrete discussion of these approaches.

3.1. Keyword Conversion Method

In our scheme, we need to convert documents and queries into vectors. We adopt the famous TF−IDF word weighting algorithm to create vectors for documents and queries. The information of term frequency (TF) is applied to create vector representations of

documents, and the inverse document frequency (IDF) knowledge is utilized to construct vector representations of queries. We give the concrete conversion approach as follows.

(1) The method extracts keywords in the dataset, and builds a dictionary $DIC = \{w_1, w_2, \ldots, w_N\}$, where w_t is an unique keyword in DIC and $t \in [1, N]$.

(2) For each document, f_i, associated with a keyword set, $W_i = \{w_{i1}, w_{i2}, \ldots, w_{in_i}\}$, the method first creates a $TF-$vector $\overrightarrow{W_i} = \{x_{i1}, x_{i2}, \ldots, x_{iN}\}$ for f_i. Based on Equation (1), the method then sets $x_{it} = TF_{w_{ij}}$ when $w_{ij} = w_t$, where $t \in [1, N]$, $i \in [1, d]$ and $j \in [1, n_i]$.

$$TF_{w_{ij}} = \frac{1 + ln(r_{w_{ij}})}{\sqrt{\sum_{w_{ij} \in W_i} (1 + ln(r_{w_{ij}}))^2}} \tag{1}$$

The number of repetitions of w_{ij} in the document, f_i, is denoted by $r_{w_{ij}}$ in Equation (1).

(3) For a query, $Q = \{q_1, q_2, \ldots, q_m\}$, the method first constructs a $IDF-$vector $\overrightarrow{Q} = \{v_1, v_2, \ldots, v_N\}$. After this, based on Equation (2), the method sets $v_t = IDF_{q_j}$ when $q_j = w_t$, where $t \in [1, N]$ and $j \in [1, m]$.

$$IDF_{q_j} = ln(1 + \frac{n}{n_{q_j}}) \tag{2}$$

In Equation (2), n_{q_j} is the number of documents that contain the keyword q_j in the dataset.

Based on the $TF-$vector, $\overrightarrow{W_i}$ and the IDF vector $\overrightarrow{Q}$, the relevance score between f_i and Q can be calculated by the following Equation (3):

$$Score(f_i, Q) = \overrightarrow{W_i} \cdot \overrightarrow{Q}. \tag{3}$$

We can obtain the most relevant documents based on relevance scores between documents and queries.

3.2. Approach for Index Building

For building the index, we first utilize the k-means clustering algorithm to divide the entire corpus into several document sets. Then, we provide a tree-building algorithm to produce an index tree for each document set. Finally, all index trees are merged to create the corpus's plaintext index. The detailed algorithms for these steps are given as follows.

3.2.1. Dataset Division Method

By using the keyword conversion method, each document, f_i, in F corresponds to a vector $\overrightarrow{W_i}$. Given the vector set $\overrightarrow{W_1}, \overrightarrow{W_2}, \ldots, \overrightarrow{W_d}$, we apply a well-known clustering algorithm $k - means$ to split the dataset F. The concrete division method is given in Algorithm 1.

Algorithm 1 first applies the $k - means$ algorithm to partition the dataset into k document clusters $C = \{c'_1, c'_2, \ldots, c'_k\}$, where the documents in each cluster are semantically relevant. Since the number of documents in each cluster is distinct, for the sake of security, Algorithm 1 then appends some fake documents to each cluster to ensure that each cluster has the same number of documents. Finally, Algorithm 1 outputs k clusters $C = \{c_1, c_2, \ldots, c_k\}$, where $c_i = \{f_{i1}, f_{i2}, \ldots, f_{i\phi}\}$, f_{ij} is a document in c_i and ϕ is the number of documents in each cluster. We build the index of the dataset F based on the partition result C.

Algorithm 1 Dataset division method.

Input: A vector set $\overrightarrow{W_1}, \overrightarrow{W_2}, \ldots, \overrightarrow{W_d}$ for the dataset F, the number of document clusters (k) that users want to produce.
Output: k document clusters $C = \{c_1, c_2, \ldots, c_k\}$.

1: Inputs $\overrightarrow{W_1}, \overrightarrow{W_2}, \ldots, \overrightarrow{W_d}$ to the $k - means$ algorithm, and obtains k document clusters $C' = \{c'_1, c'_2, \ldots, c'_k\}$;
2: Let $MaxLen$ be the maximum value of $|c'_1|, |c'_2|, \ldots, |c'_k|$, where $|c'_i|$ is the number of documents in the cluster c'_i and $i \in [1, k]$;
3: **for** each $i \in [1, k]$ **do**
4: Computes $GapLen = MaxLen - |c'_i|$.
5: **if** $GapLen > 0$ **then**
6: Constructs $GapLen$ fake documents whose TF vector is set to be zero vector;
7: Combines these $GapLen$ fake documents and c'_i to create a new cluster, c_i;
8: **end if**
9: **end for**
10: **return** $C = \{c_1, c_2, \ldots, c_k\}$;

3.2.2. Method for Building the Plaintext Index

Like some previous SSE schemes [8,9], we take advantage of the binary balanced tree as the index structure. For the sake of clarity, we define the data structure of the tree node u as: $u =< ID, \overrightarrow{u}, P_l, P_r, FID >$, where ID is the identity of u; $\overrightarrow{u}$ is the vector representation of u; P_l and P_r are two pointers which point to u's left and right children, respectively; and FID is the identity of a document if u is a leaf node. According to this formal definition, we propose the method for building the plaintext index. For each cluster, c_i, in C, to create the index tree, T_i, of c_i, we first convert the documents of the cluster, c_i, to leaf nodes. Specifically, for each document f_{ij} in c_i, we set $u_{ij}.ID = GenID()$, $u_{ij}.\overrightarrow{u} = \overrightarrow{W_{ij}}$, $u_{ij}.P_l = u_{ij}.P_r = NULL$, $u_{ij}.FID$ be the identifier of f_{ij}, where $GenID()$ is a function that can generate an unique ID for the tree node, and $\overrightarrow{W_{ij}}$ is the TF vector for documents f_{ij}, $i \in [1, k]$, and $j \in [1, \phi]$. After conversion, we can obtain a leaf node set, $l_i = \{u_{i1}, u_{i2}, \ldots, u_{i\phi}\}$, for the cluster, c_i.

Based on the leaf node set, l_i, of the cluster, c_i, we propose the index tree building algorithm for c_i in Algorithm 2.

From Algorithm 2, l_i initially contains all the leaf nodes of the cluster, c_i. We need to construct internal nodes of the index tree in a bottom–up manner based on l_i. Let $|l_i|$ be the number of nodes in l_i. If l_i contains only one node, this means that this node is the root node of the index tree of c_i. Otherwise, we should use two nodes in l_i to create a parent node. More specifically, let $l_i[t]$ and $l_i[t+1]$ be two nodes in l_i, a parent node u of these two nodes is built as follows.

The ID of u is generated by running $GenID()$. The two pointers of u are pointing to $l_i[t]$ and $l_i[t+1]$, respectively. For the vector of $\overrightarrow{u}$, $\overrightarrow{u}[j]$ is the maximum of $\overrightarrow{l_i[t]}[j]$ and $\overrightarrow{l_i[t+1]}[j]$, where $\overrightarrow{u}[j]$, $\overrightarrow{l_i[t]}[j]$, and $\overrightarrow{l_i[t+1]}[j]$ represent values of j-th dimension of $\overrightarrow{u}$, $\overrightarrow{l_i[t]}$, and $\overrightarrow{l_i[t+1]}$, respectively. By calling the function BuildTree(l_i) recursively, the plaintext index tree, T_i, for the cluster, c_i, can be constructed.

After building the index tree for each cluster, we use all the index trees as the index for the entire dataset. The index for the entire dataset is denoted by $Ind = \{r_1, r_2, \ldots r_k\}$, where r_i is the root node of T_i and $i \in [1, k]$. For the sake of clarity, we give an example to illustrate the process of index building.

Algorithm 2 The algorithm for building index tree for the cluster, c_i, declared by BuildTree(l_i)

Input: The leaf node set l_i of the cluster, c_i.
Output: The plaintext index tree, T_i, for the cluster, c_i.

```
if |l_i| == 1 then
    return l_i;
    \\This is the root node of the tree.
end if
Initializes an empty set TempNodeSet;
Sets s = |l_i|, t = 0;
while t < s do
    if t + 1 < s then
        Creates a parent node u for two nodes l_i[t] and l_i[t + 1], where u.ID = GenID(),
u.P_l = l_i[t], u.P_r = l_i[t + 1], u.FID = NULL and u⃗[j] = Max(l_i⃗[t][j], l_i⃗[t + 1][j]).
        Inserts u to TempNodeSet;
    else
        Inserts l_i[t] to TempNodeSet;
    end if
    i = i + 2;
end while
l_i = BuildTree(TempNodeSet);
\\recursively calls BuildTree.
return l_i;
```

Example 1. *An example of building the index for a document set* $F = \{f_1, f_2, \ldots, f_{12}\}$ *is illustrated in Figure 2. From Figure 2, after using clustering algorithm, we suppose that F is divided into three clusters* $c_1 = \{f_1, f_4, f_7, f_{11}\}$, $c_2 = \{f_2, f_3, f_8, f_{12}\}$, *and* $c_3 = \{f_5, f_6, f_9, f_{10}\}$. *For each cluster,* c_i, *we first convert each document in* c_i *into a leaf node. Then, based on these leaf nodes, we construct the internal nodes in a bottom–up manner using Algorithm* 2. *Finally, we combine these three index tree as the plaintext index of the document set F.*

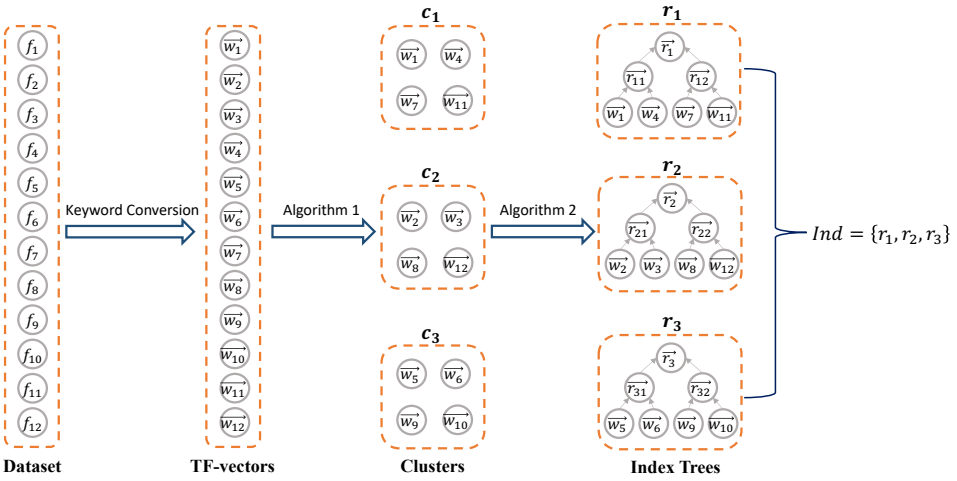

Figure 2. An example of the index building process.

3.3. Approach for Index Search

For a keyword query, Q, we adopt the keyword conversion method to transform Q into an IDF vector $\vec{Q}$. Given the index $Ind = \{r_1, r_2, \ldots r_k\}$ of the dataset, for each r_i, we will compute the relevant score between $\vec{Q}$ and $\vec{r_i}$, according to Equation (3), where $\vec{r_i}$ is the vector for the root node r_i and $i \in [1, k]$. We choose t root nodes with high correlation scores and search the index trees associated with these root nodes, where t is set by data users. Suppose that the selected root nodes are $\{r'_1, r'_2, \ldots, r'_t\}$, the search algorithm for each index tree with the root, r'_i, is described as follows, where $i \in [1, t]$.

By running Algorithm 3, we can obtain a new document set $DS = Rlist_1 \cup Rlist_2 \cup \ldots \cup Rlist_t$, where $Rlist_i$ is the search result of querying the index tree r'_i, where $i \in [1, t]$. We find θ documents with the highest correlation score from DS as query results.

Algorithm 3 The algorithm for search the index tree, declared by SearchIndexTree($\vec{Q}$, u, $RList$)

Input: An IDF vector $\overrightarrow{Q}$ of the query Q, an index tree of the root node r'_i and an empty result list $RList$.
Output: $RList$ containing documents with θ maximum relevant scores.

1: **if** u is an internal node **then**
2: SearchIndexTree($\overrightarrow{Q}$, $u.P_l$, $RList$);
3: SearchIndexTree($\overrightarrow{Q}$, $u.P_r$, $RList$);
4: **else**
5: **if** $\overrightarrow{u} \cdot \overrightarrow{Q}$>$\theta$-th score **then**
6: Deletes the element holding the smallest relevance score in $RList$;
7: Inserts a new element $< Score(u, Q), u.FID >$ in the $Rlist$, and updates the θ-th score;
8: **end if**
9: **end if**

Example 2. *In this example, we assume that the search aim is to find the documents with the top two relevance scores. From Figure 3, the entire search process consists of three parts. Firstly, we transform the query, Q, into an IDF vector* $\overrightarrow{Q}$*, and then calculate the relevance score between the query, Q, and the three index tree roots,* $\{r_1, r_2, r_3\}$*. Suppose that we retrieve only two index trees whose root nodes are most relevant to the query, Q. According to the calculation results, index tree* 1 *and tree* 3 *are chosen as our retrieval targets. Secondly, we apply Algorithm 3 to perform the retrieval operation on the index tree and obtain* f_1, f_7 *and* f_6, f_{10} *as the result of the query on index trees* 1 *and* 3*, respectively. Thirdly, we combine the query results of index tree* 1 *and tree* 3 *and return the two documents* f_1, f_{10} *with the highest correlation scores as the final query results.*

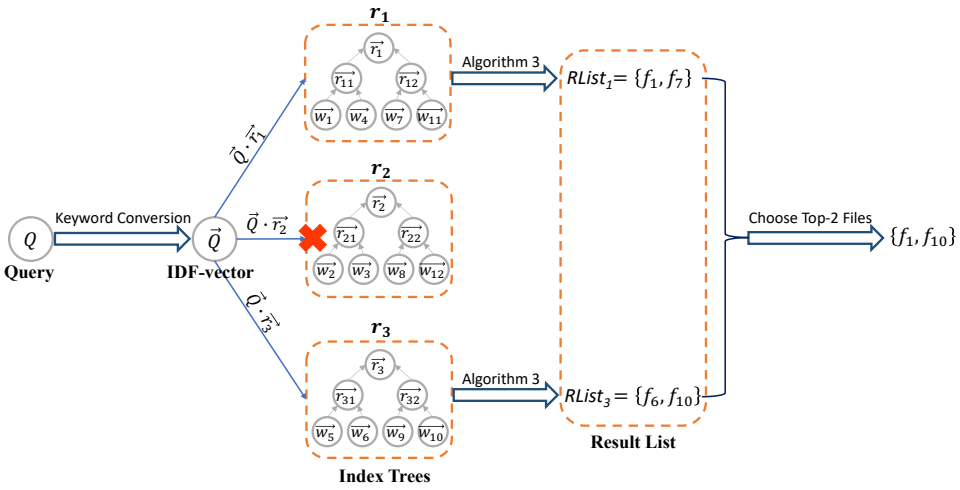

Figure 3. An example of the search process.

4. Proposed Scheme

In the last section, we introduced the construction and retrieval methods for the plaintext index. In this section, we utilize the ASPE scheme to encrypt the index and give the concrete construction method of our F-SSE-RS scheme. After this, the dynamic update approach and the security analysis for our scheme are also presented.

4.1. Construction of F-SSE-RS

According to the system model introduced in Section 2.1, the F-SSE-RS scheme consists of four algorithms: *KeyGen*, *IndexBuild*, *TrapdoorGen*, and *search*. The *KeyGen* and *IndexBuild* algorithms are executed by the data owner to generate the secrete key and create the encrypted index, respectively. The data user performs the *TrapdoorGen* algorithm to generate the encrypted trapdoor and transmits it to the cloud server. When catching the trapdoor, the cloud server runs the *search* algorithm to make the keyword query and returns the search result to the data user. The detailed construction of F-SSE-RS is given as follows.

- **KeyGen (γ)**: Taking a security parameter, γ, as an input, this algorithm chooses two random invertible matrices, M_1, M_2, whose dimension are $(N+L) \times (N+L)$, and a vector, S, whose dimension is $N+L$. Then, it sets the secret key sk as $\{M_1, M_2, S\}$ and outputs the sk to authorized data users.
- **IndexBuild (sk, F)**: Given a document set F, this algorithm first partitions F into k document subsets $\{c_1, c_2, \ldots, c_k\}$ using the data division method. For each document set, c_j, this algorithm adopts Algorithm 2 to generate an index tree T_j for c_j, where $j \in [1, k]$. Then, this algorithm encrypts the index tree, T_j. The encryption process starts from the root node, and each node is encrypted using a sequential traversal method. More precisely, for a node $u =< ID, \overrightarrow{u}, P_l, P_r, FID >$, the algorithm extends the N-dimension vector $\overrightarrow{u}$ into a $(N+L)$-dimension vector $\overrightarrow{u_E}$, in which the value of $\overrightarrow{u_E}[i]$ is set to be $\overrightarrow{u}[i]$ when $i \in [1, N]$, and the value of $\overrightarrow{u_E}[i]$ is set as a random number, ϵ_i, when $i \in [N+1, N+L]$. After the extension process, two random vectors, $\{\overrightarrow{u_E}', \overrightarrow{u_E}''\}$, of $\overrightarrow{u_E}$ can be created by using the following equations.

$$\left\{ \begin{array}{l} \overrightarrow{u_E}'[i] + \overrightarrow{u_E}''[i] = \overrightarrow{u_E}[i], \ \ if\ S[i] = 0; \\ \overrightarrow{u_E}'[i] = \overrightarrow{u_E}''[i] = \overrightarrow{u_E}[i], \ \ if\ S[i] = 1. \end{array} \right\} i \in [1, N+L].$$

  After encrypting each node in the index tree T_j, the algorithm generates the encrypted index tree E_{T_j}, where each encrypted node E_u of u can be expressed as $E_u =< ID, M_1^T \overrightarrow{u_E}', M_2^T \overrightarrow{u_E}'', P_l, P_r, FID >$ Finally, after encrypting all the index trees, the algorithm outputs the encrypted index $E_{Ind} = \{E_{T_1}, E_{T_2}, \ldots, E_{T_k}\}$.
- **TrapdoorGen (sk,Q)**: Given a query, Q, the algorithm first transforms Q into an IDF vector $\overrightarrow{Q}$ using the keyword conversion method given in Section 3.1. Then, this algorithm extends the N-dimension vector $\overrightarrow{Q}$ into a $(N+L)$-dimension vector $\overrightarrow{Q_E}$, where each $\overrightarrow{Q_E}[i]$ is set to be $\overrightarrow{Q}[i]$ when $i \in [1, N]$ and each $\overrightarrow{Q_E}[i]$ is set to be 0 or 1 randomly when $i \in [N+1, N+L]$. After this, this algorithm generates two random vectors, $\{\overrightarrow{Q_E}', \overrightarrow{Q_E}''\}$, according to the following equations.

$$\left\{ \begin{array}{l} \overrightarrow{Q_E}'[i] + \overrightarrow{Q_E}''[i] = \overrightarrow{Q_E}[i], \ \ if\ S[i] = 1; \\ \overrightarrow{Q_E}'[i] = \overrightarrow{Q_E}''[i] = \overrightarrow{Q_E}[i], \ \ if\ S[i] = 0. \end{array} \right\} i \in [1, N+L].$$

  Finally, this algorithm outputs $T_Q = \{M_1^{-1} \overrightarrow{Q_E}', M_2^{-1} \overrightarrow{Q_E}''\}$ as the trapdoor for Q.
- **Search (T_Q, E_{Ind})**: Given the trapdoor, T_Q, for each encrypted tree, E_{T_i}, this algorithm computes the relevant score, rs_i, between the encrypted root node r_i of E_{T_i} and T_Q, where $i \in [1, k]$. Suppose that $\{rs_{\rho 1}, rs_{\rho 2}, \ldots, rs_{\rho t}\}$ are the top-t correlation scores, the search algorithm performs the traversal search on these encrypted trees $\{E_{T_{\rho 1}}, E_{T_{\rho 2}}, \ldots, E_{T_{\rho t}}\}$, where $\{\rho 1, \rho 2, \ldots, \rho t\} \subset \{1, 2, \ldots, k\}$. For each $j \in [1, t]$, this algorithm searches the encrypted tree $E_{T_{\rho j}}$ according to Algorithm 3. In the search

process, for an encrypted tree node $E_u =< ID, M_1^T\overrightarrow{u_E}', M_2^T\overrightarrow{u_E}'', P_l, P_r, FID >$ and the trapdoor $T_Q = \{M_1^{-1}\overrightarrow{Q_E}', M_2^{-1}\overrightarrow{Q_E}''\}$, this algorithm can compute:

$$\begin{aligned}(M_1^T\overrightarrow{u_E}' \cdot M_1^{-1}\overrightarrow{Q_E}') + (M_2^T\overrightarrow{u_E}'' \cdot M_2^{-1}\overrightarrow{Q_E}'') &= \overrightarrow{u_E}' \cdot \overrightarrow{Q_E}' + \overrightarrow{u_E}'' \cdot \overrightarrow{Q_E}'' \\ &= \overrightarrow{u_E} \cdot \overrightarrow{Q_E} \\ &= Score(u, Q)\end{aligned} \tag{4}$$

According to Equation (4), the computation result between E_u and T_Q is the same as that between the plaintext u and Q. Therefore, the search algorithm can employ Algorithm 3 to perform the sorting search in the encrypted state. After finishing the query on the encrypted tree $E_{T_{\rho j}}$, a result set $RList_{\rho j}$ on $E_{T_{\rho j}}$ can be obtained. Finally, this algorithm figures out the θ documents with the highest scores from $RList_{\rho 1} \cup RList_{\rho 2} \cup \ldots \cup RList_{\rho t}$ and return them to the user as query results.

4.2. Dynamic Update Operations

In general, in addition to the above search requirement, our scheme also needs to satisfy the user's requirements for adding, deleting, and modifying documents. Therefore, we require three additional approaches to the scheme to support the above dynamic update operations. Since the encrypted index of the scheme is based on the balanced binary tree, we can implement these update operations by dynamically adding and deleting tree nodes. Inspired by the methods given in [8,9], the three dynamic update methods on F-SSE-RS are as follows.

- Deletion: When DO wants to delete the document f from the index, DO first determines which tree in the index f exists in. Then, DO locates the position information about the leaf node of f in that index tree. Finally, DO sends the location information to CS, which can null the node based on the location information to achieve the deletion operation.
- Addition: When DO wants to add a document f to the index, DO first transforms f's keywords into a TF vector using the keyword conversion method and constructs a leaf node about f with its TF vector. Subsequently, using the TF vector, DO finds the index tree whose root node is the most semantic similar to f in the index, and locates a leaf node marked as invalid in that tree. Then, DO replaces this invalid node with a leaf node of f and updates the vector of all internal nodes on the path from the root of the tree to this leaf node. Finally, DO encrypts all the changed nodes and sends them to CS together with their corresponding position information. When CS receives these nodes, CS replaces the relevant nodes based on the position information to implement the insertion operation. In addition, if there are no leaf nodes marked as invalid in the index tree, DO can add multiple invalid nodes to the index tree and update the index tree. After that, DO encrypts the modified tree nodes and sends their location information to CS. According to this location information, CS updates the index tree to realize the file addition operation.
- Modification: If DO wants to modify a file, then DO first locates the leaf node corresponding to that file and replaces the semantic vector for the leaf node with the newer vector. Then, DO updates all the nodes on the path from the root of the tree to that leaf node based on the modified vector of the leaf node. Finally, DO encrypts the contents of all nodes to be changed and sends their location information together to CS. When CS receives these nodes, it replaces the old nodes according to the location information to perform the update operation.

Note that the above dynamic operations all require that DO has a plaintext index stored locally. The advantage of this is that by updating the plaintext index locally, DO can obtain the information about the location and content of index updates faster. In addition, during the update process, DO encrypts the content information to be updated in the index and sends the corresponding location information to CS in plaintext, so that CS can

finish modifying the index without understanding the updated content. Although the local storage method has additional space overhead, it improves the update efficiency and security of the scheme.

4.3. Security Analysis

In this subsection, we analyze the security of the proposed F-SSE-RS scheme based on the privacy requirements introduced in Section 2.3.

- *Document and index privacy:* In the F-SSE-RS scheme, the confidentiality of the document content is guaranteed by a traditional symmetric secret key encryption scheme, such as AES. The index in the F-SSE-RS scheme is a combination of multiple index trees, and the content of each node in the index tree is cryptographically protected using the ASPE scheme. Because AES and ASPE are provably secure under known ciphertext models, the plaintext contents hidden in the documents and indices cannot be inferred by an attacker. So, we argue that the privacy of documents and indices is protected well.
- *Trapdoor unlinkability:* The trapdoor-generation algorithm of the proposed scheme is probabilistic, which is manifested in the following two aspects. (1) The semantic vector $\overrightarrow{Q}$ of the query Q is enlarged into an extension vector $\overrightarrow{Q_E}$ before generating the trapdoor, and even the same two queries can be enlarged into different extension vectors; (2) in the " TrapdoorGen" algorithm, the query vector $\overrightarrow{Q_E}$ is partitioned into two parts randomly. Based on the above two points, we can conclude that the same two queries can be encrypted into different trapdoors, so the proposed scheme can satisfy the requirement of trapdoor unlinkability.
- *Keyword privacy:* Under the known ciphertext model, the attacker cannot infer the keyword information from the index and trapdoor since the F-SSE-RS scheme utilizes the ASPE scheme to encrypt the index and trapdoor. However, in the known background model, CS can use the document–word frequency to perform statistical attacks and then infer the keywords embedded in the index and trapdoors. For the statistical attack in the known background model, our scheme extends the keyword vectors $\overrightarrow{u}$ and $\overrightarrow{Q}$ in the index and trapdoor into $\overrightarrow{u_E}$ and $\overrightarrow{Q_E}$, respectively. Specifically, for each extended dimension of $\overrightarrow{u_E}$, the scheme randomly selects a number ϵ_i, while for each extended dimension of $\overrightarrow{Q_E}$, the scheme randomly selects a number 0 or 1. This approach allows the query results to be masked by the randomness of $\sum \epsilon_i$. Since the number of extended dimensions is L, the probability that two $\sum \epsilon_i$ have the same value is only $\frac{1}{2^L}$. Therefore, when L increases, the query results will be more influenced by $\sum \epsilon_i$, bringing the result that the privacy of keywords increases but the search accuracy decreases. Therefore, by adjusting L, we can make a tradeoff between precision and privacy in practical applications. The analysis of the tradeoff between precision and privacy can be found in [8].

5. Performance Evaluation

In this section, we evaluate the proposed F-SSE-RS scheme theoretically and experimentally and give a detailed experiment to quantify the space–time efficiency of the scheme. We implemented the proposed scheme in Python and tested it on a real dataset, i.e., Enron e-mail datasets [28]. In addition, our experimental runtime environment includes an Intel(R) Core(TM) i7 CPU whose frequency is 2.90 GHz and 16 GB of RAM. To illustrate the advantages of the proposed scheme, we compare it with two similar previous schemes in terms of the time complexity of index construction, trapdoor generation and searching, and the space complexity of indexes and trapdoors. For convenience, we denote the schemes proposed in [8,9] as Xia16 and Guo19, respectively. In addition, we also conduct experiments on the accuracy of these schemes to demonstrate the merits of the proposed schemes more comprehensively.

5.1. Efficiency of Index Building

In the index-building phase, the proposed scheme splits d documents into k document clusters, each of which contains nearly d/k documents on average. For each document cluster, we construct an index tree that contains $2d/k$ nodes. Since each node contains a TF vector of length $N + L$, the time cost of encrypting a node is $O(d(N + L)^2)$, where the encryption operation mainly considers two multiplication operations between the $(N + L) \times (N + L)$-invertible matrix and the $N + L$-dimensional TF vector. Considering that the k index trees contain a total of $2d$ tree nodes, it can be deduced that the index building time of the proposed scheme is $O(d(N + L)^2)$. Moreover, since the index of the proposed scheme contains a total of $2D$ nodes and each node contains a TF vector of length $N + L$, the index storage consumption of the proposed scheme is $O(d(N + L))$. For Xia16, since its index also contains $2D$ tree nodes, the index building time of Xia16 is $O(d(N + L)^2)$ and the storage space is also $O(d(N + L))$. For Guo19, its leaf nodes are constructed in the same way as Xia16, but the vectors of the internal nodes of its index tree are compressed using the bloom filter technique. Thus, its index building time is $O(d(N + L)^2) + O(d(\alpha)^2)$, where α is the length of the bloom filter and $\alpha << N$. Because the index of Guo19 contains two different storage methods, the storage space of its scheme is $O(d(N + L)) + O(d\alpha)$. Since the index of Guo19 exists in two different vector forms, the storage space of its scheme is $O(d(N + L)) + O(d\alpha)$.

As shown in Figures 4 and 5, the index building time of Xia16, Guo19, and F-SSE-RS schemes are all squared with N (Figure 4) and linear with d (Figure 5). Specifically, when d = 1000 and N = 10,000, the index building times for Xia16, Guo19, and the proposed scheme are 886 s, 444 s, and 902 s, respectively. It is observed that the index generation time for Guo19 is half of that for Xia16 and the proposed scheme. The reason why the index building time of Guo19 is smaller than the other two schemes is that the vector dimension of its internal node is shorter. In additional, the index building time of the proposed scheme is slightly longer than that of Xia16. The reason that the index building time of the proposed scheme is longer than that of Xia16 is that our scheme has one more step of clustering operation. All the above experimental results are consistent with the theoretical analysis.

5.2. Efficiency of Trapdoor Generation

In the trapdoor-generation phase, the proposed scheme first converts the query Q into an IDF vector of dimension $N + L$, and then encrypts this vector using the ASPE scheme. Therefore, the trapdoor generation time of the proposed scheme is $O((N + L)^2)$. For Xia16, its trapdoor generation method is the same as our scheme, so the trapdoor generation time of Xia16 is also $O((N + L)^2)$. According to the above analysis, it is clear that the trapdoor storage costs of both the proposed scheme and xia16 are $O(N + L)$. For Guo19, since the internal nodes and leaf nodes are constructed by using the bloom filter vector and TF vector, respectively, its trapdoor can be seen as a binary tuple (BF_Q, TF_Q), where BF_Q and TF_Q are used to query the internal nodes and leaf nodes, respectively. Based on the above analysis, the trapdoor generation time of Guo19 is $O(N + L) + O(\alpha)$, and the trapdoor storage consumption is also $O(N + L) + O(\alpha)$.

As shown in Figure 4, the index generation time of Xia16, Guo19, and F-SSE-RS schemes are all squared with N. In particular, when d = 1000 and N = 10,000, the trapdoor generation times for Xia16, Guo19, and the proposed scheme are 440 ms, 455 ms, and 438 ms, respectively. It can be seen that the trapdoor generation time of Guo19 is more than the other two schemes. The trapdoor generation time of Guo19 is larger than the other two schemes since it has to encrypt two vectors, one for querying internal nodes and one for querying leaf nodes. Besides, the trapdoor generation time of Xia16 is the same as the trapdoor generation time of F-SSE-RS. All the above experimental results are consistent with the theoretical analysis.

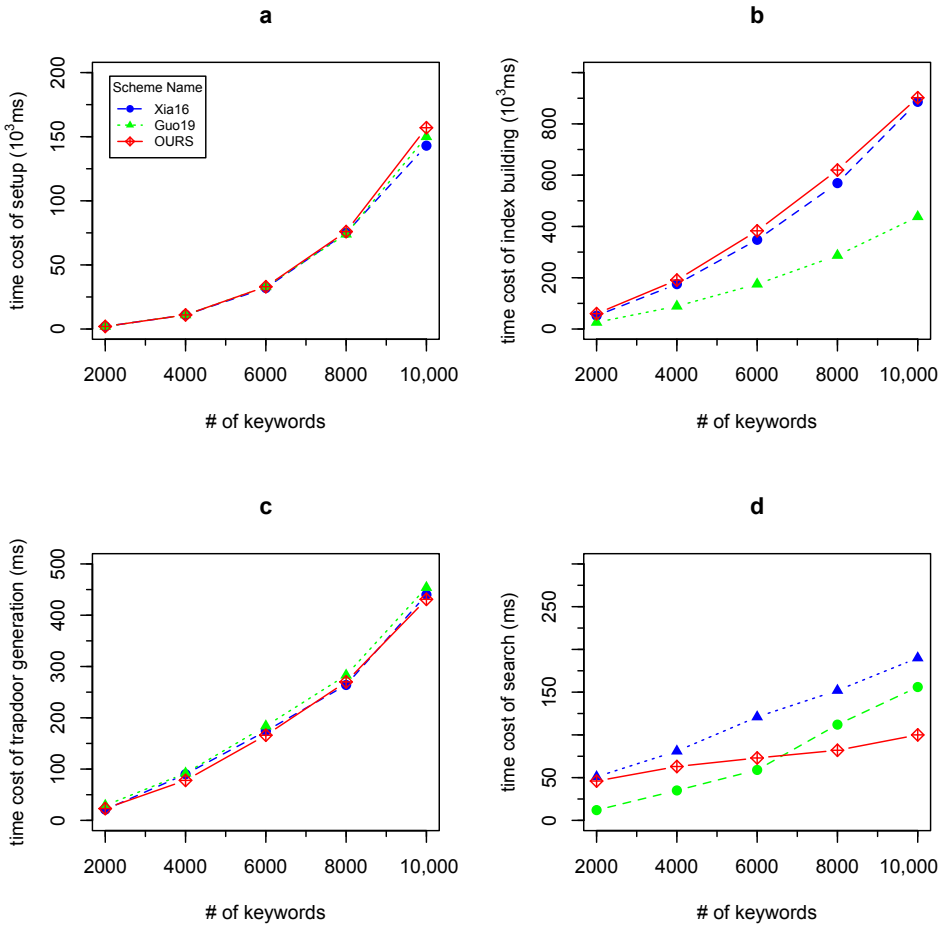

Figure 4. Impact of N on the time cost of setup (**a**), index building (**b**), trapdoor generation (**c**) and search (**d**) (N = {2000; 4000; 6000; 8000; 10,000}; d = 1000; k = 5).

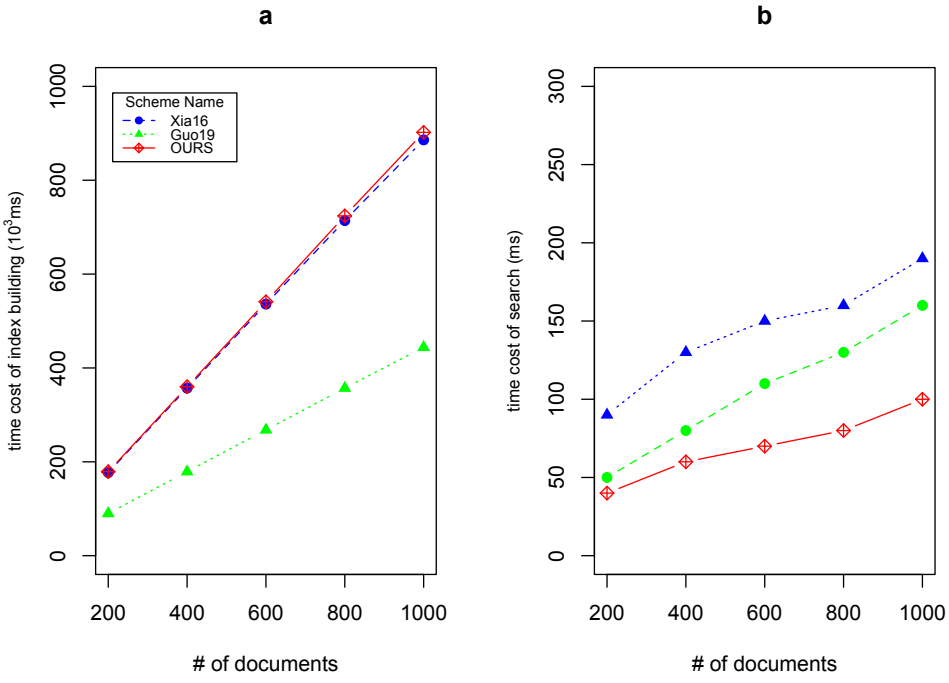

Figure 5. Impact of d on the time cost of index building (**a**) and search (**b**) (d = {200; 400; 600; 800; 1000}; N = 10,000; k = 5).

5.3. Efficiency of Search

In the search phase, because the index of the proposed scheme contains k index trees and the height of each index tree is $log_2 d/k$, the search time of each index tree is $O(log_2 d/k(N+L))$, where $N+L$ is the length of the vector contained in the internal node. In addition, when the search operation reaches the leaf nodes, the similarity calculation will be performed. The time consumption of similarity calculation for each index tree is $O(N+L)$ since the dimension of the $TF-$vector is $N+L$. Based on the above analysis, assuming that we select t most relevant index trees for querying, the search time of the proposed scheme is $O(tlog_2 d/k(N+L)) + O(N+L)$. For Xia16, it has to search the index tree with height $log_2 d$. Its query time is $O(log_2 d(N+L)) + O(N+L)$. For Guo19, its query time is $O(log_2 d\alpha) + O(N+L)$ since the vector length of the internal node is α.

As shown in Figures 4 and 5, the query times of the Xia16, Guo19, and F-SSE-RS schemes are linear with N and sublinear with d. Concretely, when d = 1000 and N = 10,000, the search times for Xia16, Guo 19, and F-SSE-RS are 98 ms, 162 ms, and 193 ms, respectively. Based on the experiment result, the search time for the proposed scheme is two-thirds of that for Guo19 and half of that for Xia16. The proposed scheme has the highest query efficiency due to the lower depth of the index tree and the smaller number of queried nodes, which is consistent with the theoretical analysis.

5.4. Accuracy

Our scheme selects only a few of the most relevant index trees for querying, which will affect the accuracy of the search results. To quantify this impact, we use the "precision definition" proposed in [7] to measure the impact of accuracy. The "precision definition" is depicted as $p = \theta/\theta'$, where θ' is the number of real top-θ files returned by CS. For clarity, we design an experiment to test the relationship between search time and query accuracy of F-SSE-RS. Concretely, we construct an index consisting of 5 index trees and select the most relevant t index trees for querying, where $t \in \{1,2,3,4,5\}$. Figure 6 shows the comparison results among the proposed scheme, Guo19, and Xia16. From Figure 6, we can find that the query accuracy of F-SSE-RS is decreasing as t reduces, but the search time is also decreasing substantially. Specifically, for every loss of approximately 10% query accuracy, there is a reduction of approximately 20% search time. In summary, according to Figure 6, compared with Xia16, F-SSE-RS improves search efficiency while not compromising query accuracy as much as possible. Compared to Guo19, the proposed scheme guarantees similar query accuracy while using less search time.

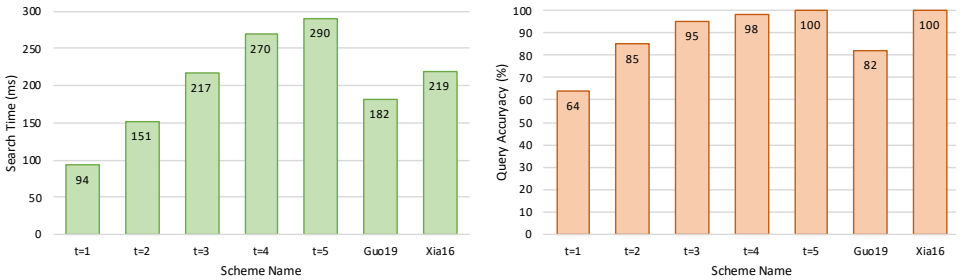

Figure 6. Impact of t on the search time and query accuracy (t = {1, 2, 3, 4, 5}; d = 1000; N = 10,000; k = 5).

5.5. Discussion

Based on the above theoretical analysis and experimental results, we can find that the proposed scheme has good flexibility compared to Xia16. Specifically, we can adjust the value of t to make a certain compromise between query accuracy and search time. Furthermore, compared with Guo19, our scheme has better query accuracy and search time, except for the longer index-building time. In real-time applications, users generally care more about search time and query accuracy, so our scheme will be more practical.

In addition, the index of the proposed scheme consists of multiple index trees, so we can accelerate the query process using parallel computing methods. Whenever DU performs a keyword search, DU sends a query trapdoor to CS. Then, CS will utilize the trapdoor to execute the search operation on multiple index trees in parallel. Each task independently performs a keyword search and adds the results to the final result set. Finally, CS returns the result set to DU. By adopting this method, the search efficiency can be significantly improved. Since the cloud platform has powerful computing power, we believe that the parallel strategy is very suitable for the proposed scheme.

The proposed scheme can be applied to cloud-based communication systems, such as wireless IoT systems [29], E-Healthcare systems [30], and personalized search systems [2]. Taking the personalized search system as an example, the users can use their terminal devices to send their encrypted query information to the cloud, and the cloud server can find the points of interest near the users and related to their queries through secure retrieval and mark them on their cloud devices. This kind of application is characterized by the real-time nature of user queries, and the fast query capability of this solution can better serve such users. In addition, the present solution has good flexibility between real-time use and accuracy. Specifically, the user can dynamically adjust the parameter t and can choose whether real-time use or accuracy is the priority. This customization setting can allow users to have a superior query experience.

6. Conclusions

In this paper, we utilize a clustering algorithm to divide the document set into multiple document clusters and index each document cluster using a binary balanced tree. When a keyword query is performed, the search algorithm retrieves only the index tree that is most semantically related to the query keyword, which effectively improves the query efficiency. By adopting an ASPE scheme to encrypt the index and query, we propose an F-SSE-RS scheme. This scheme can support ranked search on encrypted data and is secure under the known background model.

Furthermore, we give a detailed theoretical and experimental analysis. This analysis indicates that the query efficiency of the proposed scheme is sublinearly with the number of documents. In addition, our scheme has better query efficiency without compromising too much query accuracy and has better flexibility than other typical similar schemes. Thus, we believe that the proposed scheme has high practicality. Although this paper improves the query efficiency by eliminating some irrelevant documents through clustering methods, it still loses some query precision. Therefore, the future work of this paper is to further improve query accuracy while ensuring query efficiency. In addition, the proposed scheme currently supports only textual queries, while many existing cloud-based applications need to support both spatial and textual queries. Therefore, another extension work of this paper is to construct efficient searchable symmetric encryption schemes supporting spatial data queries.

Author Contributions: Conceptualization, W.H. and Y.Z.; data curation, Y.Z. and Y.L.; formal analysis, W.H. and Y.Z.; funding acquisition, W.H. and Y.Z.; methodology, W.H. and Y.Z.; software, Y.Z.; validation, W.H. and Y.Z.; writing—original draft preparation, W.H. and Y.Z.; writing—review & editing, W.H. and Y.L. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the National Natural Science Foundation of China (Grant No. 61972090, 31872704), by Natural Science Foundation of Henan (Grant No. 202300410339), and by the Science and Technology Project of Henan Province (Grant No. 212102310993).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The data used to support the findings of this study is available from the website, "URL: http://www.cs.cmu.edu/~./enron/" (accessed on 19 April 2022).

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

SE	Searchable encryption.
ASPE	Asymmetric scalar-product-preserving encryption.
SSE	Searchable symmetric key encryption.
SPE	Searchable public key encryption.
TF-IDF	Term frequency-inverse document frequency.
PEKS	encryption with keyword search.
DO	Data owner.
DU	Data user.
CS	Cloud server.

References

1. Song, D.; Wagner, D.; Perrig, A. Practical techniques for searching on encrypted data. In Proceedings of the IEEE Symposium on Research in Security and Privacy, Berkeley, CA, USA, 14–17 May 2000 ; pp. 44–55.
2. Fu, Z.; Ren, K.; Shu, J.; Sun, X.; Huang, F. Enabling personalized search over encrypted outsourced data with efficiency improvement. *IEEE Trans. Parallel Distrib. Syst.* **2015**, *27*, 2546–2559. [CrossRef]
3. Sun, W.; Liu, X.; Lou, W.; Hou, Y.T.; Li, H. Catch you if you lie to me: Efficient verifiable conjunctive keyword search over large dynamic encrypted cloud data. In Proceedings of the 2015 IEEE Conference on Computer Communications (INFOCOM), Hong Kong, China, 26 April–1 May 2015; pp. 2110–2118.
4. Boneh, D.; Di Crescenzo, G.; Ostrovsky, R.; Persiano, G. Public key encryption with keyword search. In Proceedings of the International Conference on the Theory and Applications of Cryptographic Techniques, Interlaken, Switzerland, 2–6 May 2004; pp. 506–522.
5. Zhang, Y.; Li, Y.; Wang, Y. Secure and Efficient Searchable Public Key Encryption for Resource Constrained Environment Based on Pairings under Prime Order Group. *Secur. Commun. Netw.* **2019**, *2019*, 1–14. [CrossRef]
6. Miao, Y.; Tong, Q.; Deng, R.; Choo, K.K.R.; Liu, X.; Li, H. Verifiable searchable encryption framework against insider keyword-guessing attack in cloud storage. *IEEE Trans. Cloud Comput.* **2020**, 1–14. [CrossRef]
7. Cao, N.; Wang, C.; Li, M.; Ren, K.; Lou, W. Privacy-preserving multi-keyword ranked search over encrypted cloud data. *IEEE Trans. Parallel Distrib. Syst.* **2013**, *25*, 222–233. [CrossRef]
8. Xia, Z.; Wang, X.; Sun, X.; Wang, Q. A Secure and Dynamic Multi-Keyword Ranked Search Scheme over Encrypted Cloud Data. *IEEE Trans. Parallel Distrib. Syst.* **2016**, *27*, 340–352. [CrossRef]
9. Guo, C.; Zhuang, R.; Chang, C.C.; Yuan, Q. Dynamic multi-keyword ranked search based on bloom filter over encrypted cloud data. *IEEE Access* **2019**, *7*, 35826–35837. [CrossRef]
10. Wong, W.K.; Cheung, D.W.; Kao, B.; Mamoulis, N. Secure kNN computation on encrypted databases. In Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data, Providence, RI, USA, 29 June–2 July 2009; pp. 139–152.
11. Goh, E.J. Secure indexes. *IACR Cryptol. EPrint Arch.* **2003**, *2003*, 216.
12. Wang, B.; Li, M.; Wang, H. Geometric range search on encrypted spatial data. *IEEE Trans. Inf. Forensics Secur.* **2015**, *11*, 704–719. [CrossRef]
13. Xu, G.; Li, H.; Dai, Y.; Yang, K.; Lin, X. Enabling efficient and geometric range query with access control over encrypted spatial data. *IEEE Trans. Inf. Forensics Secur.* **2018**, *14*, 870–885. [CrossRef]
14. Fu, Z.; Wu, X.; Guan, C.; Sun, X.; Ren, K. Toward Efficient Multi-Keyword Fuzzy Search Over Encrypted Outsourced Data With Accuracy Improvement. *IEEE Trans. Inf. Forensics Secur.* **2017**, *11*, 2706–2716. [CrossRef]
15. Kuzu, M.; Islam, M.S.; Kantarcioglu, M. Efficient similarity search over encrypted data. In Proceedings of the 2012 IEEE 28th International Conference on Data Engineering, Arlington, VA, USA, 1–5 April 2012; pp. 1156–1167.
16. Zhang, Y.; Li, Y.; Wang, Y. Efficient Searchable Symmetric Encryption Supporting Dynamic Multikeyword Ranked Search. *Secur. Commun. Netw.* **2020**, *2020*, 1–16. [CrossRef]
17. Wang, C.; Cao, N.; Ren, K.; Lou, W. Enabling Secure and Efficient Ranked Keyword Search over Outsourced Cloud Data. *IEEE Trans. Parallel Distrib. Syst.* **2012**, *23*, 1467–1479. [CrossRef]
18. Shao, J.; Lu, R.; Guan, Y.; Wei, G. Achieve Efficient and Verifiable Conjunctive and Fuzzy Queries over Encrypted Data in Cloud. *IEEE Trans. Serv. Comput.* **2020**, *15*, 124–137. [CrossRef]
19. Wang, X.; Ma, J.; Liu, X.; Deng, R.H.; Miao, Y.; Zhu, D.; Ma, Z. Search me in the dark: Privacy-preserving boolean range query over encrypted spatial data. In Proceedings of the IEEE INFOCOM 2020-IEEE Conference on Computer Communications, Toronto, ON, Canada, 6–9 July 2020; pp. 2253–2262.
20. Guo, C.; Chen, X.; Jie, Y.; Fu, Z.; Li, M.; Feng, B. Dynamic multi-phrase ranked search over encrypted data with symmetric searchable encryption. *IEEE Trans. Serv. Comput.* **2020**, *13*, 1034–1044. [CrossRef]
21. Park, D.J.; Kim, K.; Lee, P.J. Public key encryption with conjunctive field keyword search. In Proceedings of the International Workshop on Information Security Applications, Jeju Island, Korea, 23–25 August 2004; pp. 73–86.

22. Katz, J.; Sahai, A.; Waters, B. Predicate encryption supporting disjunctions, polynomial equations, and inner products. In Proceedings of the Annual International Conference on the Theory and Applications of Cryptographic Techniques, Istanbul, Turkey, 13–17 April 2008; pp. 146–162.
23. Xu, P.; Tang, S.; Xu, P.; Wu, Q.; Hu, H.; Susilo, W. Practical multi-keyword and boolean search over encrypted e-mail in cloud server. *IEEE Trans. Serv. Comput.* **2019**, *14*, 1877–1889. [CrossRef]
24. Miao, Y.; Liu, X.; Choo, K.K.R.; Deng, R.H.; Li, J.; Li, H.; Ma, J. Privacy-preserving attribute-based keyword search in shared multi-owner setting. *IEEE Trans. Dependable Secur. Comput.* **2021**, *18*, 1080–1094. [CrossRef]
25. Xu, P.; He, S.; Wang, W.; Susilo, W.; Jin, H. Lightweight searchable public-key encryption for cloud-assisted wireless sensor networks. *IEEE Trans. Ind. Inform.* **2017**, *14*, 3712–3723. [CrossRef]
26. Zhang, Y.; Wang, Y.; Li, Y. Searchable Public Key Encryption Supporting Semantic Multi-Keywords Search. *IEEE Access* **2019**, *7*, 122078–122090. [CrossRef]
27. Dhandabani, R.; Periyasamy, S.S.; Padma, T.; Sangaiah, A.K. Six-face cubical key encryption and decryption based on product cipher using hybridisation and Rubik's cubes. *IET Netw.* **2018**, *7*, 313–320. [CrossRef]
28. Cohen, W.W. Enron E-Mail Dataset. Available online: Http://www.cs.cmu.edu/~./enron/ (accessed on 19 April 2022).
29. Sangaiah, A.K.; Javadpour, A.; Ja'fari, F.; Pinto, P.; Ahmadi, H.; Zhang, W. CL-MLSP: The design of detection mechanism for sinkhole attacks in smart cities. *Microprocess. Microsyst.* **2022**, *90*, 104504. [CrossRef]
30. Zhang, J.; Liang, X.; Zhou, F.; Li, B.; Li, Y. TYLER, a fast method that accurately predicts cyclin-dependent proteins by using computation-based motifs and sequence-derived features. *Math. Biosci. Eng.* **2021**, *18*, 6410–6429. [PubMed]