information

MDPI

*Article*

# Efficient and Expressive Search Scheme over Encrypted Electronic Medical Records

**Xiaopei Yang [1], Yu Zhang [1,*], Yifan Wang [2] and Yin Li [3]**

[1] School of Computer and Information Technology, Xinyang Normal University, Xinyang 464031, China; xiaopei94@xynu.edu.cn
[2] Department of Computer Science, Wayne State University, 42 W Warren Ave, Detroit, MI 48202, USA; yifan.wang2@wayne.edu
[3] School of Cyberspace Security, Dongguan University of Technology, Dongguan 523820, China; liyin@dgut.edu.cn
* Correspondence: zhangyu86@xynu.edu.cn

**Abstract:** In recent years, there has been rapid development in computer technology, leading to an increasing number of medical systems utilizing electronic medical records (EMRs) to store their clinical data. Because EMRs are very private, healthcare institutions usually encrypt these data before transferring them to cloud servers. A technique known as searchable encryption (SE) can be used by healthcare institutions to encrypt EMR data. This technique enables searching within the encrypted data without the need for decryption. However, most existing SE schemes only support keyword or range searches, which are highly inadequate for EMR data as they contain both textual and digital content. To address this issue, we have developed a novel searchable symmetric encryption scheme called SSE-RK, which is specifically designed to support both range and keyword searches, and it is easily applicable to EMR data. We accomplish this by creating a conversion technique that turns keywords and ranges into vectors. These vectors are then used to construct index tree building and search algorithms that enable simultaneous range and keyword searches. We encrypt the index tree using a secure K-Nearest Neighbor technique, which results in an effective SSE-RK approach with a search complexity that is quicker than a linear approach. Theoretical and experimental study further demonstrates that our proposed scheme surpasses previous similar schemes in terms of efficiency. Formal security analysis demonstrates that SSE-RK protects privacy for both data and queries during the search process. Consequently, it holds significant potential for a wide range of applications in EMR data. Overall, our SSE-RK scheme, which offers improved functionality and efficiency while protecting the privacy of EMR data, generally solves the shortcomings of the current SE schemes.

**Keywords:** searchable symmetric encryption; electronic medical record; keyword search; range search; search over encrypted data

check for updates

**Citation:** Yang, X.; Zhang, Y.; Wang, Y.; Li, Y. Efficient and Expressive Search Scheme over Encrypted Electronic Medical Records. *Information* **2023**, *14*, 643. https://doi.org/10.3390/info14120643

Academic Editors: Jose de Vasconcelos, Hugo Barbosa and Carla Cordeiro

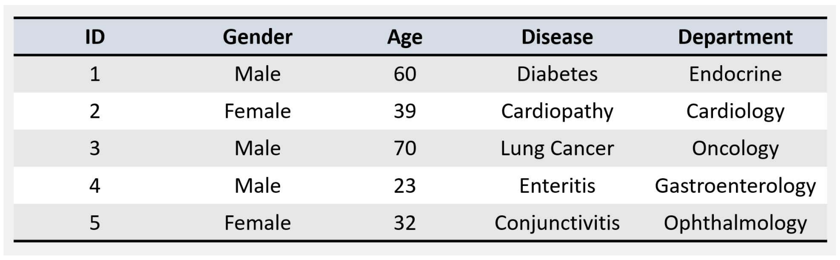
Received: 19 October 2023
Revised: 25 November 2023
Accepted: 29 November 2023
Published: 30 November 2023

## 1. Introduction

Electronic medical records use electronic devices to preserve and manage digital clinical medical records, thus replacing traditional handwritten ones. Over the past few years, with the continued development of information technology, an increasing number of medical systems are using EMRs as routine storage means. The large number of EMRs will entail large management costs for healthcare organizations. To solve this issue, EMRs can be outsourced to cloud computing service systems that have powerful storage and computing capabilities. Since medical data are highly private, medical institutions usually need to encrypt EMR data before uploading them to cloud servers to protect patient privacy. However, this protection method brings great inconvenience to the EMR retrieval operation. A simple way is to download all EMR data stored on the cloud platform to the user side and then retrieve them locally. However, this approach will cause great transmission consumption. To improve search efficiency, we can use searchable encryption

(SE) technology to encrypt these documents. This encryption technology can retrieve encrypted EMRs without decrypting them, and it can return the target documents to the querying user while protecting data security.

**Motivation.** SE allows data users to retrieve encrypted documents that are stored on cloud servers by utilizing an encrypted token without decrypting the documents. Thus far, most SE schemes support either multi-keywords search [1–3] or range search [4–6]. But, in a medical system, data users will perform a query containing both digital and textual fields. For an electronic medical record [7], it may contain both digital values and keywords. As shown in Figure 1, age and ID are numeric fields, while gender, disease, and department are keyword fields. Moreover, the user's query also contains both range and keywords content, e.g., age $\in$ [18, 45] AND disease $\in$ (diabetes, enteritis). If an SE scheme supporting only keyword search or range search is used to implement searching over encrypted EMR data, two EMR systems will be maintained: one that contains only text fields, and the other that contains only numeric fields. This not only increases the time and space complexities of the search process, but also reveals more intermediate information. Considering such an actual demand, it is necessary to build a SE scheme that can support range and keyword searches simultaneously.



| ID | Gender | Age | Disease | Department |
|---|---|---|---|---|
| 1 | Male | 60 | Diabetes | Endocrine |
| 2 | Female | 39 | Cardiopathy | Cardiology |
| 3 | Male | 70 | Lung Cancer | Oncology |
| 4 | Male | 23 | Enteritis | Gastroenterology |
| 5 | Female | 32 | Conjunctivitis | Ophthalmology |

**Figure 1.** An example of an electronic medical record.

Recently, two SE schemes [8,9] were proposed to satisfy the above practical need. In [8], Miao et al. proposed a conversion method that can transform digital points and keywords in each document to a vector representation. Using a secure K-Nearest Neighbor (KNN) algorithm to protect vector confidentiality, they presented an encryption scheme that can support range and keyword search simultaneously. Later, Wang et al. presented a SE scheme supporting spatial keyword queries. Their solution can support arbitrary geometry, as well as keyword, queries, which can be applied to realize both textual keywords and digital range queries. In their scheme, by utilizing the techniques of gray code and bloom filter, files and queries can be transformed into a series of "0-1-*" strings. For privacy preserving purposes, the obtained strings are encrypted by applying the symmetric-key hidden vector encryption (SHVE) scheme [10].

However, the above two schemes still have two issues that are causes for concern. First, only integer range query is supported in these two schemes. The reason why these two schemes cannot support decimal range searches stems from the specificity of their core methods. The scheme given in [8] uses the modulo operation to support multi-dimensional range queries. Since the modulo operation is an integer operation, this scheme can only support integer range queries. The scheme presented in [9] adopts the "gray code" encoding method to convert ranges into "0-1" bit strings. This encoding method only supports integer ranges as a legal input. However, in an electronic medical record, the range query containing decimals is very common, such as white blood cell count, blood glucose level, tumor size, etc. To overcome this shortcoming, two range encoding methods have to be given to implement range search, which can support range searches with decimals. Second, the efficiency of these two schemes could be still improved. More precisely, the scheme proposed in [8] adopts an index structure with a linear search time complexity, while the scheme in [9] will enumerate many gray codes to perform a range search. To address this issue, we take advantage of the tree-based index structure to construct an efficient SSE scheme that supports range and keyword queries simultaneously.

**Contributions.** In the interest of clarity, we list the major contributions of this article.

(1) We propose a keyword conversion method that can transform a collection of keywords into a vector. Furthermore, based on the characteristics of the range query, we give two ways through which to convert a range query into a vector. These vectors can be utilized to perform both range and keyword searches efficiently.

(2) We design an index tree construction algorithm to speed up the query process. The internal node of the index tree contains only one range vector, and the leaf node contains a small number of vectors for both points and keywords. Based on the index tree, through an efficient prune algorithm, the query time of the proposed scheme is sublinear to the number of documents.

(3) Through using the secure KNN scheme [11] to encrypt the index tree and query, we propose an SSE scheme that can support both range and keyword queries (SSE-RK), which can be applied in searching electronic medical records efficiently.

To show the security of the proposed scheme, we will give a detailed security analysis of SSE-RK. In addition, we conducted quantitative experiments on SSE-RK on a medical dataset. The experimental results show that the proposed scheme can effectively perform ciphertext retrieval on EMR data.

**Related Work.** According to the characteristics of its secret key, searchable encryption (SE) schemes are usually divided into symmetrical and asymmetrical approaches.

For searchable symmetric encryption (SSE), the data uploader and the authorized query user hold the same key. Song et al. [12] designed the first SSE scheme, which only supports single keyword queries. Goh then developed a more formal definition of security for SSE, and they utilized Bloom Filter technology to build an SSE scheme [13] that supports multi-keyword queries. Subsequently, many works [14–17] have focused on the efficiency improvement of SSE schemes. However, these schemes will return all the matched documents without sorting, which requires a great deal of computing and communication costs. To address this problem, two SSE schemes that support ranked search were proposed in [18,19]. A ranked search scheme will return the $k$ most relevant documents based on a given similarity evaluation criterion. As a result, the solution will significantly reduce communication and storage consumption. In response to the problem that the scheme in [18,19] only supports single-keyword queries, Cao et al. proposed a ranked search scheme that supports multi-keyword queries [20]. However, the query efficiency of this scheme is linearly related to the number of documents due to its using a forward index. To improve the query efficiency, tree index-based schemes were proposed in [21,22]. The search time complexity of these schemes is sublinear to the number of documents. Recently, Liu et al. [23] presented a scheme for protecting spatial data privacy and user query privacy in location-based service providers (LBSP). The scheme uses Hilbert curves and an SSE algorithm as the basic building blocks to achieve accurate range queries. By utilizing a special inverted index structure and an oblivious memory access algorithm, an SSE scheme that supports single-keyword range queries with efficient performance was proposed in [24]. Zheng et al. [25] proposed an efficient and privacy-preserving exact set similarity search scheme under a single cloud server using symmetric key predicate encryption and $B^+$-tree indexing. By combining attribute-based encryption (ABE) with fog computing architecture, a secure and efficient fine-grained searchable data sharing and management scheme in IoT-based smart healthcare systems was introduced in [26]. This scheme can achieve secure and efficient fine-grained searchable data sharing and management. In addition, there are many works devoted to constructing SSE schemes with more expressive queries, such as semantic search [27,28] and fuzzy search [29,30], which greatly improve the flexibility of ciphertext retrieval schemes. Considering that the attributes of a document will contain both digital and keyword content, two SSE schemes [8,9] that support range and keyword queries simultaneously were proposed to meet this practical requirement. But, the efficiency and functionality of these schemes can be improved. Thus, we designed a scheme to solve these problems in this paper.

Searchable asymmetric encryption, also known as searchable public key encryption (SPE), contains a pair of secret keys in which the public key is used to encrypt the data and the private key is used for privacy queries. Boneh et al. first proposed the definition of SPE, and they created an SPE scheme that supports single keyword retrieval [31]. To support conjunctive keyword searches, Park et al. proposed a public key encryption with conjunctive keyword search (PECK) scheme [32]. To support disjunctive keyword searches, Katz et al. proposed a predicate encryption scheme [33]. To support both conjunctive and disjunctive keyword retrieval, Zhang and Lu proposed a public key encryption with a conjunctive and disjunctive keyword search (PECDK) scheme [34], which is based on the inner product encryption scheme [35]. To increase the security of SPE, an SPE that resists keyword guessing attacks [36] and an SPE with access control capability [37] have also received more attention.

**Organization.** The structure of this paper is as follows. The formal definition of the system and security model will be given in Section 2, and the design goals of the proposed scheme are introduced. Various conversion methods, index generation, and retrieval algorithms will be given in Section 3. Section 4 will give the concrete scheme and the security analysis of the scheme. Theoretical and experimental analysis will be given in Section 5. Section 6 concludes the paper.

## 2. Problem Formulation

First, we define the system model of the SSE-RK scheme. Then, the threat model faced by the SSE-RK scheme is presented. Finally, we summarize the design goals of the SSE-RK scheme. For the sake of clarity, we summarize the notation of this paper in Table 1.

**Table 1.** Notation descriptions in the SSE-RK scheme.

| | |
|---|---|
| $F$ | A set of documents $\{f_1, f_2, \ldots, f_d\}$. |
| $d$ | The number of documents in $F$ |
| $DIC = \{dic_1, dic_2, \ldots, dic_N\}$ | The dictionary of a corpus. |
| $W_i = \{w_{i1}, w_{i2}, \ldots, w_{i\|W_i\|}\}$ | The keyword set for the document $f_i$, where $i \in [1, d]$. |
| $\|W_i\|$ | The number of keywords in $W_i$, and $i \in [1, N]$. |
| $w_{ij}$ | The $j$-th keywords in $W_i$, and $i \in [1, N]$, $j \in [1, \|W_i\|]$. |
| $\overrightarrow{W_i}$ | The vector representation for $W_i$. |
| $p_i = \{x_{i1}, x_{i2}, \ldots, x_{im}\}$ | The multi-dimension point for a document $f_i$, where $i \in [1, d]$. |
| $m$ | The dimension of a multi-dimension point. |
| $u$ | A node in the index tree. |
| $I_u$ | The encrypted node $u$ in the index. |
| $I_T$ | The encrypted index tree of $F$. |
| $u_P$ | The vector set for a multi-dimension point in a leaf node $u$. |
| $\overrightarrow{u_W}$ | The vector representation for a keyword set in a leaf node $u$. |
| $\overrightarrow{u_R}$ | The vector representation for a range in an internal node $u$. |
| $Q = (Q_R, Q_W)$ | A query tuple. |
| $Q_R$ | A query range. |
| $Q_W$ | A query keyword set. |
| $\overrightarrow{Q_R}$ | The vector representation used to search internal nodes. |
| $Q_P$ | The vector set for search leaf nodes. |
| $\overrightarrow{Q_W}$ | The vector representation for making keyword search. |
| $T_Q$ | The trapdoor of the query $Q$. |

### *2.1. System Model*

As shown in Figure 2, similar with most SSE schemes [12–16], the system model of SSE-KR consists of three different roles: data owner (DO), data user (DU), and cloud server (CS). For SSE-RK, four main protocols are included: key generation, index building, trapdoor generation, and secure search. Specifically, the responsibility of the DO is to encrypt all documents, build secure indexes, and send them to a CS. The responsibility of the DU is to issue queries, i.e., generate secure trapdoors, and to send them to a CS. The CS is responsible for performing the secure search and returning the query results to the DU.

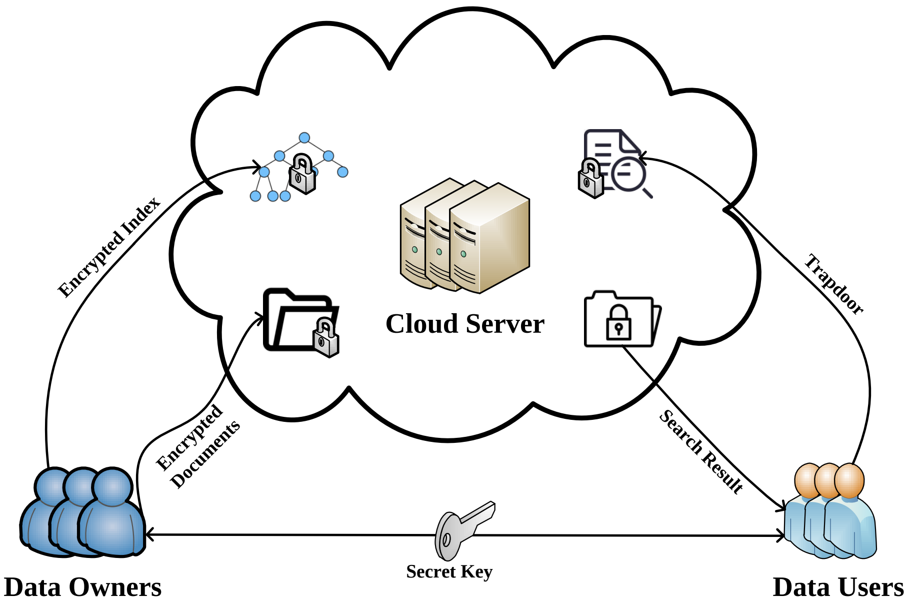


**Figure 2.** System model of SSE-RK.

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

(1) Data owner (DO). Before outsourcing a document set $F = \{f_1, f_2, \ldots, f_d\}$ to a CS, the DO first generates the secret key, then encrypts $F$ using traditional symmetric encryption algorithms (e.g., AES, etc.), and then constructs a secure index using the generated key. Finally, they upload the encrypted data and the secure index to a CS for storage. When a legitimate DU requests a query, the DO shares the secret key with the legitimate DU through authorization.

(2) Data user (DU). When an authorized DU wants to launch a query $Q$, DU generates a trapdoor using the secret key shared by DO. After this, the DU sends the trapdoor to a CS. Once the DU receives the encrypted documents back from a CS, they decrypt these documents using the secret key to recover the original plaintext.

(3) Cloud server (CS). The main function of a CS is to store files and perform retrieval. A CS stores the encrypted data and secure index uploaded by the DO. When an authorized DU uploads a trapdoor without any decryption, a CS performs a matching query on the secure index and the trapdoor, as well as returns the encrypted result of the query to the DU.

### *2.2. Threat Model*

Through this paper, like many SE schemes [19–21], we assume that the DO and DU are credible and that the CS is "honest-but-curious". This means that the CS executes algorithms of SSE-RK honestly and correctly, but it will curiously infer and analyze obtained data to learn extra private information. According to the above assumption, the two threat models introduced in [20] were considered in the proposed scheme.

- **Known ciphertext model.** Only contains information of ciphertexts, secure indexes, and trapdoors that can be obtained by a CS, which means that only ciphertext-only attacks can be performed in this model.
- **Known background model.** A CS can obtain more background knowledge, e.g., term frequency (TF)-inverse document frequency (IDF), than the aforementioned model. This information is commonly acquired from documents by statistical means. The CS can conduct the statistical attack by utilizing such information.

### *2.3. Design Goals*

Recall that our goal is to create a secure, efficient SSE scheme that supports both range and keyword searches. For the sake of clarity, we present the explicit design goals as follows.

(1) **Functionality.** The document $f_i$ of SSE-RK contains a point set $p_i = \{x_{i1}, x_{i2}, \dots, x_{im}\}$ and a keyword set $W_i = \{w_{i1}, w_{i2}, \dots, w_{i|W_i|}\}$. The query $Q$ of SSE-RK can be a hybrid of a range set $Q_R = \{[a_1, b_1], [a_2, b_2], \dots, [a_m, b_m]\}$ and a keyword query $Q_W = \{q_1, q_2, \dots, q_s\}$. The search result of the SSE-RK scheme can be ranked, thus meaning that SSE-RK only returns documents whose point $x_{ij}$ is in the query range $[a_j, b_j]$ and whose keyword set $W_i$ is strongly correlated with the query keywords $Q_W$ as the search result, where $i \in [1, d]$ and $j \in [1, s]$.

(2) **Efficiency.** The query time of the SSE-RK is sublinear to the number of documents. Specifically, the proposed scheme has better search efficiency than other similar schemes without sacrificing much index building efficiency.

(3) **Privacy preserving.** Similar to previous schemes [19–21], the SSE-RK scheme disallows CSs to obtain extra private information. This information can be inferred from ciphertexts, secure indexes, and trapdoors. More explicitly, we list the privacy requirement of SSE-RK as follows.

   - *Index and trapdoor privacy.* SSE-RK prevents CSs from inferring plaintext information that is hidden in indexes and trapdoors. That is to say, information including points, keywords, and their corresponding vectors cannot be disclosed to CSs.
   - *Trapdoor unlinkability.* In real-world scenarios, CSs sometimes receive the same query request. If a CS can easily capture two trapdoors that are generated from a single query request, an adversary can launch statistical attacks, e.g., an increase in the frequency of a certain query may indicate that the user tends to retrieve popular content, thus compromising the privacy of the query request.
   - *Keyword privacy.* CSs cannot utilize background knowledge and statistics to infer whether a trapdoor contains a particular keyword. When CSd can infer the frequency of keyword occurrences from the trapdoor, it can infer the main content of the ciphertext data.

## 3. Algorithms for Index Building and Searching

We first introduce some of the useful conversion methods used in the proposed scheme, which includes a keyword conversion approach and two range conversion methods. Then, based on these conversion methods, we present a method for building an index tree. This method consists of three steps: the construction of leaf nodes based on all documents; the construction of internal nodes based on all leaf nodes; and the use of a recursive algorithm that builds an index tree based on all nodes. Finally, the algorithm for searching the index tree is presented. A detailed description of these methods is proposed in the following sub-sections.

### *3.1. Keyword Conversion Method*

In the proposed scheme, both the document and query are converted into vectors. When a query is executed, by calculating the similarity between vectors, the documents with the highest scores are returned as the search result. In our scheme, we take advantage of a keyword conversion method based on a term weighting formula called TF−IDF to implement rank search [20]. Through adopting the TF−IDF formula, a document and a query are converted into a TF-vector and an IDF-vector, respectively. Similar to the method introduced in [28,38], we introduce the keyword conversion method adopted in SSE-RK as follows.

(1) Creating a dictionary $DIC = \{dic_1, dic_2, \ldots, dic_N\}$ by extracting keywords in the corpus, where $dic_t$ is a keyword and $t \in [1, N]$.

(2) Given a keyword set $W_i = \{w_{i1}, w_{i2}, \ldots, w_{i|W_i|}\}$, this approach first creates a zero vector $\overrightarrow{W_i} = \{x_{i1}, x_{i2}, \ldots, x_{iN}\}$. Then, it sets $x_{it} = TF_{w_{ij}}$ according to the Equation (1) if $w_{ij} = dic_t$, where $t \in [1, N]$, $i \in [1, d]$ and $j \in [1, |W_i|]$.

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

In Equation (1), $n_{w_{ij}}$ is the number of times $w_{ij}$ appears in the document $f_i$.

(3) Given a keyword set $Q_W = \{q_1, q_2, \ldots, q_s\}$, this approach first initializes a zero vector $\overrightarrow{Q_W} = \{v_1, v_2, \ldots, v_N\}$. Then, it sets $v_t = IDF_{q_j}$ according to the Equation (2) if $q_j = dic_t$, where $t \in [1, N]$ and $j \in [1, s]$.

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

The variable $n_{q_j}$ in Equation (2) represents the number of documents that contains the keyword $q_j$.

Given $\overrightarrow{W_i}$ and $\overrightarrow{Q_W}$, we can utilize Equation (3) to calculate the similarity between $f_i$ and $Q_W$.

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

We can obtain a list of documents that are most relevant to the query by taking advantage of the similarity score between each document and the query.

### *3.2. Range Conversion Methods*

For range queries, there are two frequently used operations. The first is to check whether a value $x$ is in the range of $[a, b]$; the last is to judge whether a range $[x, y]$ intersects with a range $[a, b]$. In this subsection, to adopt the vector space model mentioned previously, we present two range conversion methods to vectorize these two operations as above.

**Method** $M_1$**.** Given a value $x$ and a range $[a, b]$, we can construct Equation (4) to check whether $x \in [a, b]$.

$$\begin{aligned} f(x) &= (b - x)(x - a) \\ &= -x^2 + (b + a)x - ab \end{aligned} \tag{4}$$

Based on the root and coefficient of $f(x)$, two vectors, $\overrightarrow{x} = \{x^2, x, 1\}$ and $\overrightarrow{ab} = \{-1, a + b, -ab\}$, were created, where $\overrightarrow{x}$ and $\overrightarrow{ab}$ are for the value $x$ and the range $[a, b]$, respectively. It is easy to verify that $x \in [a, b]$ if $\overrightarrow{x} \cdot \overrightarrow{ab} >= 0$. For simplicity, we denoted this conversion method as $M_1$. $M_1$ is used to convert the operation of whether a point belongs to a range into a vector inner product operation. When constructing the leaf nodes of an index tree, we use $M_1$ to convert the multi-dimensional point of a document into a set of vectors.

**Method** $M_2$**.** Given two ranges, $[a, b]$ and $[x, y]$, if $[a, b]$ intersects with $[x, y]$, the mid value $m$ of $[a, b]$ must be in the range $[x - c, y + c]$, where $m = \frac{b+a}{2}$ and $c = \frac{b-a}{2}$. According to this property, Equation (5) was constructed.

$$\begin{aligned} f(x,y) &= (y+c-m)(m-x+c) \\ &= (y-a)(b-x) \\ &= -ab+ax+by-xy \end{aligned} \tag{5}$$

Based on Equation (5), the two vectors of $[x,y]$ and $[a,b]$ are $\overrightarrow{xy} = \{1,x,y,xy\}$ and $\overrightarrow{ab} = \{-ab,a,b,-1\}$, respectively. It can be verified that $[a,b]$ intersects with $[x,y]$ if $\overrightarrow{xy} \cdot \overrightarrow{ab} >= 0$. For simplicity, we called this conversion method $M_2$. $M_2$ was employed to convert the operation of whether two ranges intersect into a vector inner product operation. When constructing the internal nodes of an index tree, we used $M_2$ to convert the multi-dimensional range of an internal node into a vector.

## 3.3. Algorithm for Creating the Leaf Node

Since the index tree is constructed in a bottom-up manner, we built the leaf nodes first. In our scheme, each document $f_i$ contains one multi-dimension point $p_i = \{x_{i1}, x_{i2}, \ldots, x_{im}\}$ and a keyword set $W_i$. The algorithm for creating leaf nodes is given in Algorithm 1. The set of leaf nodes produced by Algorithm 1 will be used as the input to the index tree building algorithm.

The formal definition of any node $u$ on the index tree is $u =< ID, \overrightarrow{u_W}, \overrightarrow{u_R}, u_P, P_l, P_r, FID >$. $ID$ represents the identity information of $u$, which is generated by a random function $GenID()$. $\overrightarrow{u_W}$ is a vector representation of the keyword set $W_i$ associated with the leaf node, and $\overrightarrow{u_R}$ is a vector representation of the range associated with the internal node. $u_P$ is a set of vector representations of the multi-dimensional points contained in the leaf node. $P_l$ and $P_r$ are pointers to the left and right child nodes, respectively. $FID$ is the ID of the document associated with the leaf node.

The algorithm for leaf node construction is given in Algorithm 1. Specifically, for each document $f_i$ containing the tuple $(W_i, p_i)$, the algorithm runs $GenID()$ to assign a value to $u.ID$ and sets $u.FID$ to the identifier of $f_i$. Since leaf nodes have no children, both $u.P_l$ and $u.P_r$ were set to null values. Through applying the keyword conversion method introduced in Section 3.1 to $W_i$, we converted $W_i$ into a keyword vector $\overrightarrow{x_i}$ and set $\overrightarrow{u_W} = \overrightarrow{x_i}$. For the point $p_i$ of $f_i$, each value in $p_i$ was transformed into a vector by adopting the method $M_1$. More specifically, for each $x_{ij} \in p_i$, a vector $\overrightarrow{x_{ij}} = \{x_{ij}^2, x_{ij}, 1\}$ can be created. After this, $u_P$ is set to be $\{\overrightarrow{x_{i1}}, \overrightarrow{x_{i2}}, \ldots, \overrightarrow{x_{im}}\}$.

**Algorithm 1** Creating leaf nodes.

**Input:** A set of tuples $\{(W_1,p_1),(W_2,p_2),\ldots,(W_d,p_d)\}$, where $W_i$ and $p_i$ are the keyword set and multi-dimensional point for $f_i$, respectively, and $i \in [1,d]$.
**Output:** A $LeafNodeSet$ that contains all leaf nodes.

```
1: for each $i \in [1,d]$ do
2:     initializes a leaf node $u$ for $f_i$;
3:     runs GenID() to set a unique identifier for $u.ID$, assigns the identifier of $f_i$ to $u.FID$,
   and sets $u.P_l = u.P_r = NULL$.
4:     runs the keyword conversion method to transform $W_i$ to $\overrightarrow{x_i}$, and sets $\overrightarrow{u_W} = \overrightarrow{x_i}$;
5:     For each $x_{ij} \in p_i$, creates a vector $\overrightarrow{x_{ij}} = \{x_{ij}^2, x_{ij}, 1\}$, and sets $u_p = \{\overrightarrow{x_{i1}}, \overrightarrow{x_{i2}}, \ldots, \overrightarrow{x_{im}}\}$.
6:     Inserts $u$ to $LeafNodeSet$;
7: end for
8: return $LeafNodeSet$
```

## 3.4. Algorithm for Building the Index Tree

The algorithm takes a set of leaf nodes as the input and builds the internal nodes of the tree in a bottom-up manner by calling the algorithm recursively, which means that an internal node is constructed by two child nodes. As such, before putting forward the tree building algorithm, we first propose a method for constructing an internal node, and we called this method $M_3$.

**Method** $M_3$**.** In our scheme, each internal node $u$ has a set of ranges, e.g., $[x_1, y_1]$, $[x_2, y_2]$,..., $[x_m, y_m]$. Suppose that $min(\alpha, \beta)$ and $max(\alpha, \beta)$ are two simple functions, where $min(\alpha, \beta)$ and $max(\alpha, \beta)$ output the minimum and maximum values of $\alpha$ and $\beta$, respectively. For the two nodes $u'$ and $u''$, the range of the internal node (parent node) $u$ is constructed as follows.

(1) Let $u'$ and $u''$ be two leaf nodes, where the points in $u'$ and $u''$ are $p'' = \{x'_1, x'_2, \ldots, x'_m\}$ and $p'' = \{x''_1, x''_2, \ldots, x''_m\}$, respectively. For each sub-range $[x_j, y_j]$ in $u$, $x_j$ and $y_j$ are set to be $min(x'_j, x''_j)$ and $max(x'_j, x''_j)$, respectively, where $j \in [1, m]$.

(2) Let $u'$ and $u''$ be two internal nodes, where the ranges in $u'$ and $u''$ are $[x'_1, y'_1]$, $[x'_2, y'_2]$,..., $[x'_m, y'_m]$ and $[x''_1, y''_1]$, $[x''_2, y''_2], \ldots, [x''_m, y''_m]$, respectively. For each sub-range $[x_j, y_j]$ in $u$, this method sets $x_j = min(x'_j, x''_j)$ and $y_j = max(y'_j, y''_j)$, where $j \in [1, m]$.

After obtaining $u$'s range, we need to convert the range into a range vector. Given the range $[x_1, y_1]$, $[x_2, y_2]$,..., $[x_m, y_m]$ of $u$, a vector $\overrightarrow{u_R}$=$\{1, x_1, y_1, x_1y_1, 1, x_2, y_2, x_2y_2, \ldots, 1, x_m, y_m, x_my_m\}$ can be created by $M_2$, where $\{1, x_j, y_j, x_jy_j\}$ is the vector for the sub-range $[x_j, y_j]$.

Inspired by the index tree construction algorithm in [28,38], the approach for constructing the index tree is given in Algorithm 2. $LeafNodeSet$ contains a set of nodes. Each node in $LeafNodeSet$ does not have a parent node and is needed to be processed. The overall idea of the algorithm is to construct an internal node using every two nodes in $LeafNodeSet$ until there is only one node left in $LeafNodeSet$, which means that this unique node is the root of the tree. Specifically, suppose $LeafNodeSet[i]$ and $LeafNodeSet[i+1]$ are any two nodes in $LeafNodeSet$, then their parent node $u$ is created as follows. First, let $u.P_l$ and $u.P_r$ point to nodes $LeafNodeSe[i]$ and $LeafNodeSet[i+1]$, respectively, and generate unique ID for $u$ with $GenID()$; then, by taking advantage of $M_3$, create $\overrightarrow{u_R}$ based on the range vectors of $LeafNodeSet[i]$ and $LeafNodeSet[i+1]$; and finally, add $u$ to $TempNodeSet$. Note that, if $|LeafNodeSet|$ is odd, the last node in $LeafNodeSet$ will be inserted to $TempNodeSet$ directly. When the parents of all nodes in $LeafNodeSet$ have been created and added to $TempNodeSet$, Algorithm 2 will be called recursively with $TempNodeSet$ as the input until the index tree is constructed.

**Algorithm 2** The index tree building algorithm, declared by BuildIndexTree(LeafNodeSet)

**Input:** $LeafNodeSet$ including all the leaf nodes.
**Output:** An index tree $T$.

```
1: Sets $k = |LeafNodeSet|$;
2: if $k == 1$ then
3:     return $LeafNodeSet$; \\This only node is the root of the tree.
4: end if
5: Initializes an empty set $TempNodeSet$;
6: for each $i \in [1, k/2]$ do
7:     Constructs a parent node $u$ for $LeafNodeSet[2*i-1]$ and $LeafNodeSet[2*i]$;
8:     Utilizes $GenID()$ to generate an unique ID for $u.ID$;
9:     Sets $u.P_l = LeafNodeSet[2*i-1]$ and $u.P_r = LeafNodeSet[2*i]$;
10:    Generates a vector $\overrightarrow{u_R}$ for its corresponding range according to $M_3$;
11:    Inserts $u$ to $TempNodeSet$;
12: end for
13: if $k\%2 == 1$ then
14:     Inserts $LeafNodeSet[k]$ to $TempNodeSet$;
15: end if
16: $LeafNodeSet = BuildIndexTree(TempNodeSet)$; \\calls BuildIndexTree recursively.
17: return $LeafNodeSet$;
```

**Example 1.** *To better understand Algorithm 2, we show an example of the index tree construction in Figure 3, which consists of two steps. Suppose that $F = \{f_1, f_2, \ldots, f_8\}$, then it first transforms each document $f_i$ into a leaf node $u_i$ by Algorithm 2, where $i \in [1, 8]$. Concretely, for each $f_i$ that* *contains a point $p_i$ and a keyword set $W_i$, we convert $p_i$ and $W_i$ into $u_p$ and $\overrightarrow{u_W}$ by utilizing $M_1$ and the keyword conversion method, respectively. The second step is to build the index tree based on the leaf nodes from the bottom up. More specifically, we generate the range vector $\overrightarrow{u_R}$ of each internal node u from the range vectors of its two child nodes by adopting $M_3$. After these two steps, the plaintext index tree is built.*

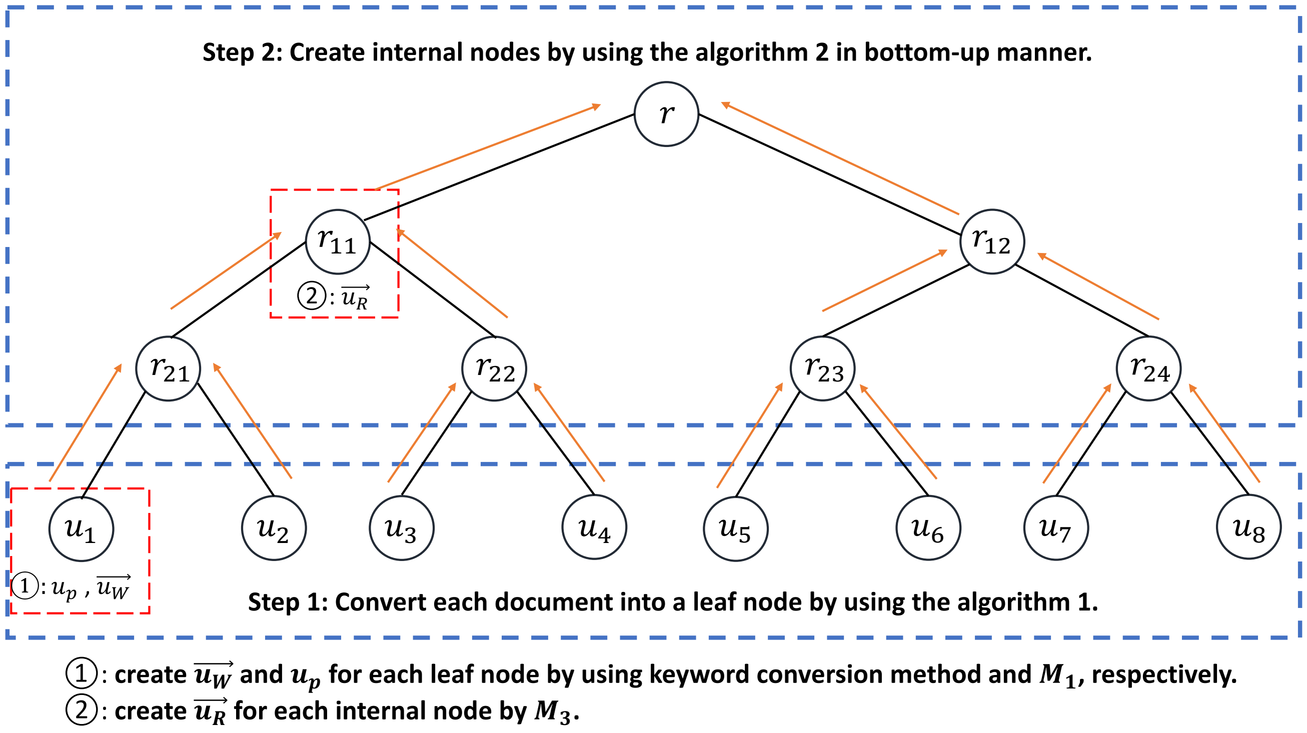

Step 2: Create internal nodes by using the algorithm 2 in bottom-up manner.
$r$
$r_{11}$
$r_{12}$
②: $\overrightarrow{u_R}$
$r_{21}$
$r_{22}$
$r_{23}$
$r_{24}$
$u_1$
$u_2$
$u_3$
$u_4$
$u_5$
$u_6$
$u_7$
$u_8$
①: $u_p$ , $\overrightarrow{u_W}$
Step 1: Convert each document into a leaf node by using the algorithm 1.
①: create $\overrightarrow{u_W}$ and $u_p$ for each leaf node by using keyword conversion method and $M_1$, respectively.
②: create $\overrightarrow{u_R}$ for each internal node by $M_3$.

**Figure 3.** An example of how to build an index tree (Algorithm 2).

### *3.5. Algorithm for Searching the Index Tree*

Before proposing the search algorithm, we first give a method for converting a query $Q = (Q_R, Q_W)$ into a query tuple that contains three elements $(\overrightarrow{Q_R}, Q_P, \overrightarrow{Q_W})$, where $Q_R$ is a query range, $Q_W$ is a query keyword set, $\overrightarrow{Q_R}$ is a vector used to search internal nodes, $Q_P$ is a group of vectors used to search leaf nodes, and $\overrightarrow{Q_W}$ is a vector for conducting a keyword search. The query transformation approach, declared by $QueryTransform$, is given as follows.

(1) Given the query range $Q_R$, based on $M_2$, each sub-range $[a_j, b_j]$ is converted into $\{-a_jb_j, a_j, b_j, -1\}$, where $j \in [1, m]$. According to this conversion, a vector $\overrightarrow{Q_R} = \{-a_1b_1, a_1, b_1, -1, -a_2b_2, a_2, b_2, -1, \ldots, -a_mb_m, a_m, b_m, -1\}$ can be created.

(2) Based on $M_1$, each sub-range $[a_j, b_j]$ of $Q_R$ is converted into $\overrightarrow{v_j} = \{-1, b_j + a_j, -a_jb_j\}$, and $Q_P$ is set to be $\{\overrightarrow{v_1}, \overrightarrow{v_2}, \ldots, \overrightarrow{v_m}\}$.

(3) Apply the keyword conversion method to transform $Q_W$ into $\vec{v}$, and set $\overrightarrow{Q_w} = \vec{v}$.

Inspired by the index tree search algorithm introduced in [28,38], the search algorithm used in SSE-RK is presented in Algorithm 3. In Algorithm 3, we used $RList$ to store the $k$ documents that are currently most relevant to the query and their corresponding similarity scores, as well as designate the $k$-score as the minimum similarity score in $RList$. Initially, $RList$ is an empty list and the $k$-score is set to a very small number. Given a query tuple $(\overrightarrow{Q_R}, Q_p, \overrightarrow{Q_W})$ of $Q$, an index tree root node $u$, and an empty result list $RList$, the goal of the index tree search algorithm is to obtain the $k$ documents that satisfy the range query and are most relevant to the query keywords. The search process is divided into two scenarios. (1) When an internal node is retrieved, the inner product between

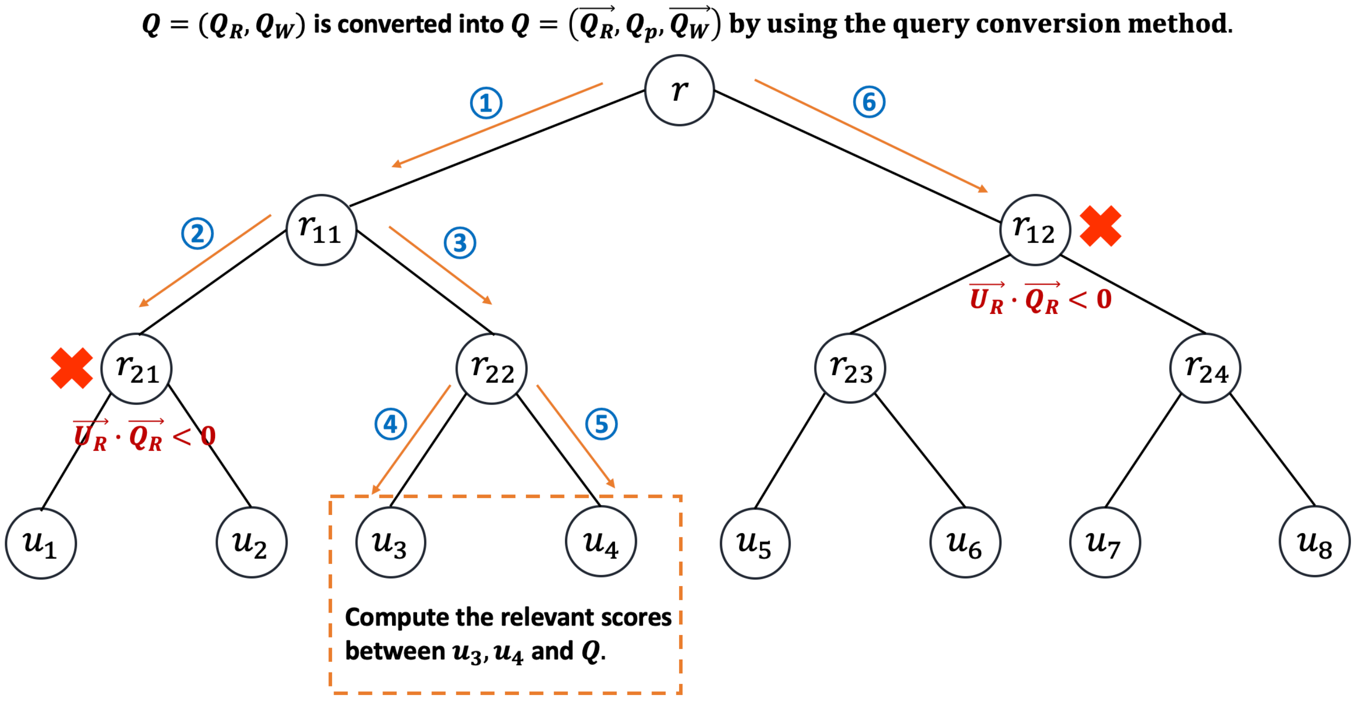
$\overrightarrow{Q_R}$ and $\overrightarrow{u_R}$ is calculated. The pruning method in our scheme is verifying whether the query range $Q_R = \{[a_1, b_1], [a_2, b_2], \ldots, [a_m, b_m]\}$ in $Q$ intersects with the range $\{[x_1, y_1], [x_2, y_2], \ldots, [x_m, y_m]\}$ in an internal node $u$. If the range in $u$ intersects with the range $Q_R$, then it must be $\overrightarrow{u_R} \cdot \overrightarrow{Q_R} > 0$, which means that the subtree of the internal node still needs to be traversed. Thus, the algorithm continues to be called for the subtree of this node. If the inner product is less than 0, then the subtree is pruned and will not be visited further. (2) When the leaf node is retrieved, then the first step is to determine whether $Q_p$ satisfies $u_p$. That is, let $u_P$ be $\{\overrightarrow{x_1}, \overrightarrow{x_2}, \ldots, \overrightarrow{x_m}\}$, and let $Q_P$ be $\{\overrightarrow{v_1}, \overrightarrow{v_2}, \ldots, \overrightarrow{v_m}\}$. Moreover, the search algorithm tests whether $\overrightarrow{x_j} \cdot \overrightarrow{v_j}$ equals 0 for all $j \in [1, m]$. If so, it represents that the multi-dimensional point $p$ implied by the leaf node belongs to the query range $Q_R$, and that it requires further computation of the correlation score between $\overrightarrow{Q_W}$ and $\overrightarrow{u_W}$. If the score is greater than the document with the smallest relevance score in the current *RList*, the document corresponding to that leaf node is added to the *RList*, and the document with the smallest relevance score in the *RList* is removed. Otherwise, the document corresponding to this leaf node is discarded.

**Algorithm 3** The index tree search algorithm, declared by SearchIndexTree($\vec{Q_R}$, $Q_P$, $\vec{Q_W}$, $u$, *RList*)

**Input:** A query tuple $(\overrightarrow{Q_R}, Q_P, \overrightarrow{Q_W})$ of query $Q$, the index tree's root node $u$, and an empty result list *RList*.
**Output:** *RList*.

```
1:  if $u$ is an internal node then
2:      if $\overrightarrow{u_R} \cdot \overrightarrow{Q_R} > 0$ then \\ Determine whether the query range $Q_R$ intersects the range $u_R$ implied by the internal node.
3:          SearchIndexTree(($\overrightarrow{Q_R}, Q_P, \overrightarrow{Q_W}$), $u.P_l$, $RList$);
4:          SearchIndexTree(($\overrightarrow{Q_R}, Q_P, \overrightarrow{Q_W}$), $u.P_r$, $RList$);
5:      else
6:          return
7:      end if
8:  else
9:      if $\overrightarrow{x_j} \cdot \overrightarrow{v_j} = 0$ for all $j \in [1, m]$ then \\ Determine whether the multi-dimensional point $p$ implied by the leaf node belongs to the query range $Q_R$.
10:         if $\overrightarrow{u_W} \cdot \overrightarrow{Q_W} > k$-score then \\ Calculate whether the correlation score between the document keywords and the query keywords is greater than the smallest score in the current $RList$.
11:             Removes the document with the smallest relevance score in the $RList$;
12:             Adds the tuple $< Score(u_W, Q_W), u.FID >$ to the $Rlist$;
13:             Sets the $k$-score to the smallest relevance score in the current $RList$;
14:         end if
15:     end if
16:     return
17: end if
```

**Example 2.** *In this example, we suppose that only the top-1 file will be returned to the data user, and Figure 4 was constructed to show the index tree search process. When using the query transformation approach, a query $Q = (Q_R, Q_W)$ is converted into a tuple $(\overrightarrow{Q_R}, Q_P, \overrightarrow{Q_W})$. According to the index tree shown in Figure 3, the search algorithm starts from the root node r and reaches the internal node $r_{11}$ first. Since the inner product between the $\overrightarrow{u_R}$ of $r_{11}$ and $\overrightarrow{Q_R}$ of Q was larger than 0, the search algorithm accessed its child nodes. Because the inner product between the $\overrightarrow{u_R}$ of $r_{21}$ and $\overrightarrow{Q_R}$ of Q was smaller than 0, then the two child nodes $u_1$ and $u_2$ of $r_{21}$ will not be reached. Since the node $r_{22}$ matches the query Q, Algorithm 3 computes the relevant scores between $u_3$ and Q, as well as adds $u_3$ to RList directly since the number of documents in the RList had not reached the upper limit. When reaching $u_4$, Algorithm 3 computes the relevant score between $u_4$ and Q, as well as compares this score to the k-score. If the relevant score between $u_4$ and Q is larger than the k-score, Algorithm 3 deletes $u_3$ from RList and adds $u_4$ to RList instead; otherwise, nothing happens. When the left subtree is checked, Algorithm 3 will detect node $r_{12}$. Since the query Q was not related to the node $r_{12}$, the subtree with $r_{12}$ as the root node would not be accessed. After this, Algorithm 3 output RList. Numbers 1–6 in Figure 4 illustrate the tree traversal process. It can be seen that subtrees* *with $r_{12}$ or $r_{21}$ as the tree root will not be visited. This pruning improves the query efficiency of the scheme.*



$Q = (Q_R, Q_W)$ is converted into $Q = (\overrightarrow{Q_R}, Q_p, \overrightarrow{Q_W})$ by using the query conversion method.
$r$
$r_{11}$
$r_{12}$
$r_{21}$
$r_{22}$
$r_{23}$
$r_{24}$
$u_1$
$u_2$
$u_3$
$u_4$
$u_5$
$u_6$
$u_7$
$u_8$
①
②
③
④
⑤
⑥
$\overrightarrow{U_R} \cdot \overrightarrow{Q_R} < 0$
$\overrightarrow{U_R} \cdot \overrightarrow{Q_R} < 0$
Compute the relevant scores between $u_3, u_4$ and $Q$.

**Figure 4.** An example of the search process (Algorithm 3).

## 4. Proposed Scheme

In this section, we used the algorithms introduced in Section 3 to construct a concrete SSE-RK scheme, as well as to perform a theoretical analysis of the security of the proposed scheme.

### 4.1. Construction of SSE-RK

According to the system model proposed in Section 2, the SSE-RK scheme first needs to create an algorithm that can generate secret keys. Secondly, for data owners and users, SSE-RK needs to build algorithms that can generate secure searchable indexes and a trapdoor. Finally, for the cloud server, SSE-RK should construct a search algorithm to enable the secure retrieval of the encrypted index. According to the above description, the SSE-RK scheme consists of four algorithms: the secret key generation algorithm, *KeyGen*; the index building algorithm, *IndexBuild*; the trapdoor generation algorithm, *TrapdoorGen*; and the secure search algorithm, *Search*. In order to better demonstrate the relationship between the roles of the system model and the four algorithms mentioned above, we constructed Figure 5 to show the interaction process between these roles. Specifically, the DO runs the *KeyGen* algorithm to generate the secret key *sk*, and then sends *sk* to the authorized DU. The DO uses the *CreatLeafNode* algorithm (Algorithm 1) and the *BuildIndexTree* algorithm (Algorithm 2) to transform the document set into an index tree. The *IndexBuild* algorithm is then used to generate the tree into a secure index, which is then sent to a CS. Whenever the DU wants to perform a query, the DU generates a trapdoor regarding $Q$ via the *TrapdoorGen* algorithm, which is then sent to a *CS*. Once the CS receives the trapdoor, it executes the *Search* algorithm and returns the search result, *RList*, to the DU. The detailed construction process of these four algorithms is given below.

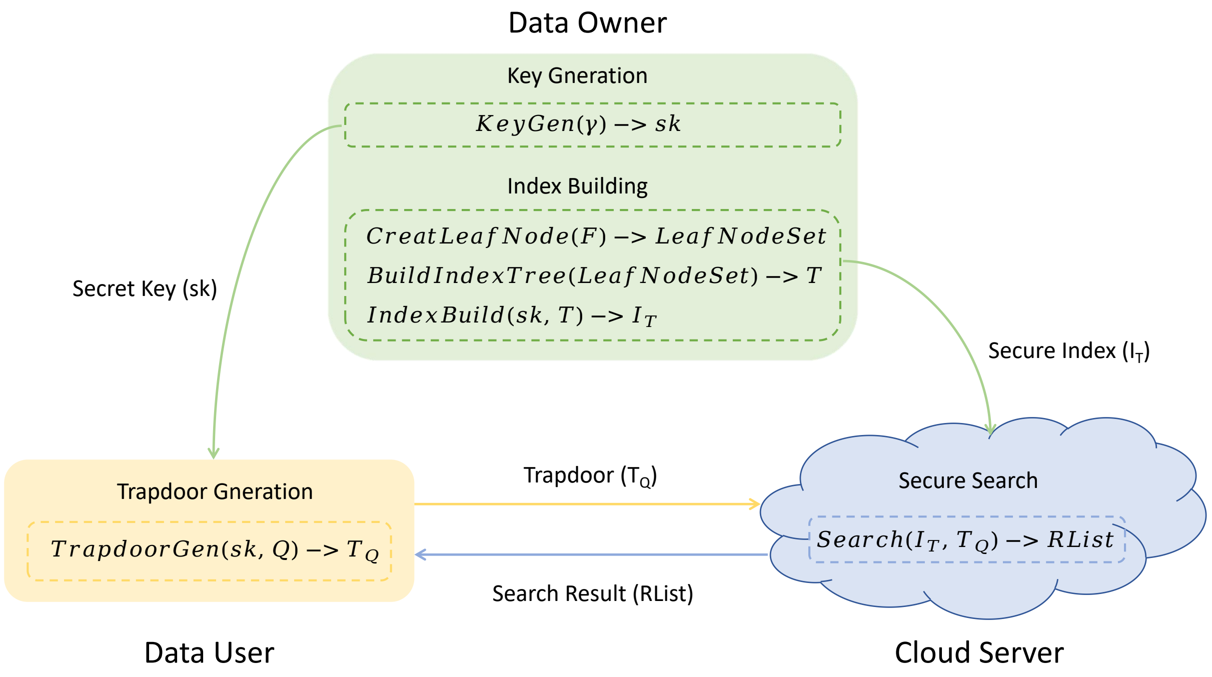


**Figure 5.** The process of interaction between the roles in the system model.

- **KeyGen**($\gamma$): Given a security parameter $\gamma$ as input, it first randomly generates two $3 \times 3$ invertible matrices $M_{11}$ and $M_{12}$; two $4m \times 4m$ invertible matrices $M_{21}$ and $M_{22}$; and two $(N+L) \times (N+L)$ invertible matrices $M_{31}$ and $M_{32}$. Then, it randomly generates three vectors $S_1, S_2$, and $S_3$, where the dimensions of $S_1, S_2$, and $S_3$ are 3, $4m$, and $(N+L)$, respectively. Finally, it outputs the secret key $sk = \{S_1, M_{11}, M_{12}, S_2, M_{21}, M_{22}, S_3, M_{31}, M_{32}\}$.
- **IndexBuild**($sk$, $F$): Given the document set $F$, it applies Algorithm 2 to build a plaintext index tree $T$, and it then encrypts $T$. The encryption process can be classified into two situations.
  (1) For each internal node $u =< ID, NULL, \overrightarrow{u_R}, NULL, P_l, P_r, NULL >$, the algorithm generates two random vectors $\{\overrightarrow{u_R}', \overrightarrow{u_R}''\}$ of $\overrightarrow{u_R}$. More precisely, if $S_2[i] = 0$, it sets $\overrightarrow{u_R}'[i] + \overrightarrow{u_R}''[i] = \overrightarrow{u_R}[i]$; if $S_2[i] = 1$, then $\{\overrightarrow{u_R}'[i], \overrightarrow{u_R}''[i]\}$ are set to two random numbers that satisfy condition $\overrightarrow{u_R}'[i] = \overrightarrow{u_R}''[i] = \overrightarrow{u_R}[i]$, where $i \in [1, 4m]$. This procedure can be represented by the following equation.

$$\left\{ \begin{array}{ll} \overrightarrow{u_R}'[i] + \overrightarrow{u_R}''[i] = \overrightarrow{u_R}[i], & if\ S_2[i] = 0; \\ \overrightarrow{u_R}'[i] = \overrightarrow{u_R}''[i] = \overrightarrow{u_R}[i], & if\ S_2[i] = 1. \end{array} \right\} i \in [1, 4m].$$

  Then, it generates the encrypted internal node $I_u =< ID,\ NULL,\ \{M_{21}^T \overrightarrow{u_R}', M_{22}^T \overrightarrow{u_R}''\}, NULL, P_l, P_r, NULL >$.
  (2) For each leaf node $u =< ID, \overrightarrow{u_W}, NULL, u_P, P_l, P_r, FID >$, the encryption process contains two steps.
    - The algorithm initializes an empty set $\hat{u_P}$. For each vector $\overrightarrow{x_j}$ in $u_P$, the algorithm generates two random vectors $\{\overrightarrow{x_j}', \overrightarrow{x_j}''\}$ of $\overrightarrow{x_j}$, where $j \in [1, m]$. Similarly, this procedure can be represented by the following equation.

$$\left\{ \begin{array}{ll} \overrightarrow{x_j}'[i] + \overrightarrow{x_j}''[i] = \overrightarrow{x_j}[i], & if\ S_1[i] = 0; \\ \overrightarrow{x_j}'[i] = \overrightarrow{x_j}''[i] = \overrightarrow{x_j}[i], & if\ S_1[i] = 1. \end{array} \right\} i \in [1, 3].$$

      After this, it adds $\{M_{11}^T \overrightarrow{x_j}', M_{12}^T \overrightarrow{x_j}''\}$ into the $\hat{u_P}$.

- For the keyword vector $\overrightarrow{u_W}$ in the leaf node, the $N$-dimension vector $\overrightarrow{u_W}$ is stretched to a $(N+L)$-dimension vector $\overrightarrow{u_{WE}}$. For $\overrightarrow{u_{WE}}$, the value of $\overrightarrow{u_{WE}}[i]$ is set to be $\overrightarrow{u_W}[i]$ when $i \in [1,N]$, and the value of $\overrightarrow{u_{WE}}[i]$ is set as a random number $\epsilon_i$ when $i \in [N+1, N+L]$. Then, the algorithm generates two random vectors $\{\overrightarrow{u_{WE}}', \overrightarrow{u_{WE}}''\}$ of $\overrightarrow{u_{WE}}$ according to the following equations.

$$\left\{\begin{array}{l} \overrightarrow{u_{WE}}'[i] + \overrightarrow{u_{WE}}''[i] = \overrightarrow{u_{WE}}[i], \;\; if\ S_3[i] = 0; \\ \overrightarrow{u_{WE}}'[i] = \overrightarrow{u_{WE}}''[i] = \overrightarrow{u_{WE}}[i], \;\; if\ S_3[i] = 1. \end{array}\right\} i \in [1, N+L].$$

After these two steps, it generates the encrypted leaf node $I_u =< ID, \{M_{31}^T\overrightarrow{u_{WE}}', M_{32}^T\overrightarrow{u_{WE}}''\}, NULL, \hat{u_P}, P_l, P_r, FID >$.

Finally, when the encryption operation is completed for each node, the algorithm outputs the encrypted index tree $I_T$.

- **TrapdoorGen**($sk$,$Q$): For a query $Q = (Q_R, Q_W)$, this algorithm applies the query conversion method introduced in Section 3.5 to generates a query tuple $(\overrightarrow{Q_R}, Q_P, \overrightarrow{Q_W})$. After this, it will encrypt the query tuple. The encryption process can be classified into three situations.

(1) For $\overrightarrow{Q_R}$, it generates two random vectors $\{\overrightarrow{Q_R}', \overrightarrow{Q_R}''\}$. This division process is similar to the index building algorithm and can still be represented by the following equation.

$$\left\{\begin{array}{l} \overrightarrow{Q_R}'[i] + \overrightarrow{Q_R}''[i] = \overrightarrow{Q_R}[i], \;\; if\ S_2[i] = 0; \\ \overrightarrow{Q_R}'[i] = \overrightarrow{Q_R}''[i] = \overrightarrow{Q_R}[i], \;\; if\ S_2[i] = 1. \end{array}\right\} i \in [1, 4m].$$

After this, it replaces $\overrightarrow{Q_R}$ with $\{M_{21}^{-1}\overrightarrow{Q_R}', M_{22}^{-1}\overrightarrow{Q_R}''\}$.

(2) The algorithm initializes an empty set $\hat{Q_P}$. For each vector $\overrightarrow{v_j}$ in $Q_P$, the algorithm generates two random vectors $\{\overrightarrow{v_j}', \overrightarrow{v_j}''\}$ of $\overrightarrow{v_j}$ according to the following equations, where $j \in [1, m]$.

$$\left\{\begin{array}{l} \overrightarrow{v_j}'[i] + \overrightarrow{v_j}''[i] = \overrightarrow{v_j}[i], \;\; if\ S_1[i] = 0; \\ \overrightarrow{v_j}'[i] = \overrightarrow{v_j}''[i] = \overrightarrow{v_j}[i], \;\; if\ S_1[i] = 1. \end{array}\right\} i \in [1, 3].$$

After this, it adds each $\{M_{11}^{-1}\overrightarrow{v_j}', M_{12}^{-1}\overrightarrow{v_j}''\}$ into $\hat{Q_P}$.

(3) The $N$-dimension vector $\overrightarrow{Q_W}$ is expanded to a $(N+L)$-dimension vector $\overrightarrow{Q_{WE}}$. For each $i \in [1, N]$, it sets $\overrightarrow{Q_{WE}}[i] = \overrightarrow{Q_W}[i]$. For each $i \in [N+1, N+L]$, it chooses a random number 0 or 1, and it sets $\overrightarrow{Q_{WE}}[i]$ to be equal to 0 or 1. Then, it adopts the following equations to create two random vectors $\{\overrightarrow{Q_{WE}}', \overrightarrow{Q_{WE}}''\}$.

$$\left\{\begin{array}{l} \overrightarrow{Q_{WE}}'[i] + \overrightarrow{Q_{WE}}''[i] = \overrightarrow{Q_{WE}}[i], \;\; if\ S_3[i] = 0; \\ \overrightarrow{Q_{WE}}'[i] = \overrightarrow{Q_{WE}}''[i] = \overrightarrow{Q_{WE}}[i], \;\; if\ S_3[i] = 1. \end{array}\right\} i \in [1, N+L].$$

After this, it replaces $\overrightarrow{Q_W}$ with $\{M_{31}^{-1}\overrightarrow{Q_{WE}}', M_{32}^{-1}\overrightarrow{Q_{WE}}''\}$.

Finally, this algorithm outputs the trapdoor $T_Q = \{\{M_{21}^{-1}\overrightarrow{Q_R}', M_{22}^{-1}\overrightarrow{Q_R}''\}, \hat{Q_P}, \{M_{31}^{-1}\overrightarrow{Q_{WE}}', M_{32}^{-1}\overrightarrow{Q_{WE}}''\}\}$ for $Q$.

- **Search** ($I_T$, $T_Q$): Given an encrypted index tree $I_T$ and a trapdoor $T_Q$, this algorithm executes the search operation in a pre-order traversal manner. When an internal node

$I_u =< ID, NULL, \{M_{21}^T\overrightarrow{u_R}', M_{22}^T\overrightarrow{u_R}''\}, NULL, P_l, P_r, NULL >$ is accessed, it computes the following:

$$\begin{aligned}(M_{21}^T\overrightarrow{u_R}' \cdot M_{21}^{-1}\overrightarrow{Q_R}') + (M_{22}^T\overrightarrow{u_R}'' \cdot M_{22}^{-1}\overrightarrow{Q_R}'') &= \overrightarrow{u_R}' \cdot \overrightarrow{Q_R}' + \overrightarrow{u_R}'' \cdot \overrightarrow{Q_R}'' \\ &= \overrightarrow{u_R} \cdot \overrightarrow{Q_R}\end{aligned} \quad (6)$$

When reaching a leaf node $I_u =< ID, \{M_{31}^T\overrightarrow{u_{WE}}', M_{32}^T\overrightarrow{u_{WE}}''\}, NULL, \hat{u_P}, P_l, P_r, FID >$, the computation process has two steps.

(1) For each $\overrightarrow{x_j}', \overrightarrow{x_j}''$ in $\hat{u_P}$ and each $\overrightarrow{v_j}', \overrightarrow{v_j}''$ in $\hat{Q_P}$, where $j \in [1, m]$, it computes the following:

$$\begin{aligned}(M_{11}^T\overrightarrow{x_j}' \cdot M_{11}^{-1}\overrightarrow{v_j}') + (M_{12}^T\overrightarrow{x_j}'' \cdot M_{12}^{-1}\overrightarrow{v_j}'') &= \overrightarrow{x_j}' \cdot \overrightarrow{v_j}' + \overrightarrow{x_j}'' \cdot \overrightarrow{v_j}'' \\ &= \overrightarrow{x_j} \cdot \overrightarrow{v_j}\end{aligned} \quad (7)$$

(2) To evaluate the relevance score, it computes the following:

$$\begin{aligned}(M_{31}^T\overrightarrow{u_{WE}}' \cdot M_{31}^{-1}\overrightarrow{Q_{WE}}') + (M_{32}^T\overrightarrow{u_{WE}}'' \cdot M_{32}^{-1}\overrightarrow{Q_{WE}}'') &= \overrightarrow{u_{WE}}' \cdot \overrightarrow{Q_{WE}}' + \overrightarrow{u_{WE}}'' \cdot \overrightarrow{Q_{WE}}'' \\ &= \overrightarrow{u_{WE}} \cdot \overrightarrow{Q_{WE}}\end{aligned} \quad (8)$$

According to the above Equations (6)–(8), the computation result between the encrypted node $I_u$ and the trapdoor $T_Q$ is identical to that between the plaintext $u$ and the query $Q$. Thus, this algorithm can take advantage of Algorithm 3 to execute a ranked search.

### 4.2. Security Analysis

As described in Section 2.3, the proposed scheme needs to satisfy three security requirements such as "Index and trapdoor privacy", "Trapdoor Unlinkability", and "Keyword privacy". In the following, we will analyze the security of the proposed solution in detail based on these three requirements.

**Index and trapdoor privacy.** For a privacy-preserving scheme, the objective is to preserve as much sensitive information about the adversary as possible while successfully obtaining the correct result. Based on the method in [39], we give the following definition before conducting the security proof.

History: According to Table 1, $F = \{f_1, f_2, \ldots, f_d\}$ is the set of documents, where $f_i$ represents the $i$-th document, $I_T$ is the index tree constructed from $F$ using the index building algorithm, and $Q_S = \{Q_1, Q_2, \ldots Q_t\}$ is the set of queries that have been executed. The history associated with $Q_S$ is defined as $H_{Q_S} = \{F, I_T, Q_S\}$.

View: View represents what can be seen by adversaries in the scheme. Specifically, we use the AES scheme to encrypt $F$. The ciphertext of $F$ is denoted as $C*$. Furthermore, we use the secure KNN scheme to encrypt the index tree and queries. The encrypted index tree and trapdoors are denoted as $I_T*$ and $T_D*$, respectively. The adversary's view is defined as $\{C*, I_T*, T_D*\}$.

Trace: Traces of history are additional information that adversaries can obtain during the execution of a scheme. It is mainly the access pattern and search pattern leaked by the user when the user makes queries on the encrypted index $I_T*$ when using the trapdoor collection $T_D*$. The access pattern is the query results that correspond to each query, while the search pattern is a matrix where, if the element in row $i$ and column $j$ of the matrix is 1, then it means that the query condition $q_i$ is the same as the query condition $q_j$.

Based on the definitions of the terms above, we give the following lemma and provide detailed steps of the proof of the lemma.

**Lemma 1.** *Given the two histories with the same trace, the proposed scheme is said to be secure if the adversaries of the probability polynomial time cannot distinguish their views.*

**Proof.** For a particular trace, if a polynomial-time simulator $S$ exists, it can generate a simulated index $I_{T_S}*$, a series of simulated trapdoors $T_{D_S}*$, and a simulated encrypted document set $C_S*$, i.e., a simulated $views_S = \{C_S*, I_{T_S}*, T_{D_S}*\}$. We say that the proposed scheme is secure if the adversary cannot distinguish the simulated $views_S$ from the real view with a non-negligible probability. Below, we give the concrete simulation procedure for the proof.

- S generates a simulated encrypted document set $C_S*$. Firstly, S generates $f_i^S \in \{0,1\}^{\{|f_i|\}}$, where $1 \leq i \leq d$ and $\{0,1\}^{\{|f_i|\}}$ are represented as a binary string of length $|f_i|$. Then, S encrypts $f_i^S$ to create $f_i^S*$ such that $|f_i^S*| = |f_i*|$, where $|f_i^S*|$ and $|f_i*|$ are the ciphertexts of $f_i^S$ and $f_i$, respectively. Finally, S outputs $C_S* = \{f_i^S*|1 \leq i \leq d\}$. Since the AES scheme is secure, it is guaranteed that $C_S*$ and $C*$ cannot be distinguished by an adversary.
- S generates the simulated trapdoor set $T_{D_S}*$. S generates $t$ query conditions, i.e., $Q' = \{Q'_1, Q'_2, ..., Q'_t\}$. For each $Q'_j$, the *TrapdoorGen* algorithm can be utilized to produce it as a trapdoor, where $1 \leq j \leq t$. Since the essence of the *TrapdoorGen* algorithm is to encrypt $Q'_j$ using the secure KNN scheme, it ensures that $T_{D_S}*$ and $T_D*$ cannot be distinguished by an adversary.
- S generates the simulated index tree $I_{T_S}*$. For each simulated document $f_i^S$, S generates the simulated multi-dimensional point $p_i^S$ and the keyword set $W_i^S$ based on the query set $Q'$. For each query $Q'_j = (Q_R^j, Q_W^j)$ in which $1 \leq j \leq t$, the $p_i^S$ generated by S needs to satisfy $p_i \in Q_R^j$ and $p_i^S \in Q_R^j$ and $Score(W_i, Q_W^j) = Score(W_i^S, Q_W^j)$. Here, $p_i$ and $W_i$ are the multi-dimensional point and keyword set of the real document $f_i$, respectively, where $1 \leq i \leq d$. S will generate a simulated index tree $I_{T_S}$ for all the the simulated points and keyword sets using the index tree building algorithm, and it will encrypt the $I_{T_S}$ using the *IndexBuild* algorithm to create an encrypted index tree $I_{T_S}*$. Since the secure KNN scheme we use is secure under a known ciphertext model, the SSE-RK scheme can guarantee the indistinguishability of $I_{T_S}*$ from $I_T*$.

By utilizing the $T_{D_S}*$ to query $I_{T_S}*$, it can be verified that the simulated $views_S$ will produce the same trace as the real view. Since the AES and secure KNN schemes are provably secure, this means that, based on the same trace, there is no probabilistic polynomial time adversary that can distinguish between the simulated $views_S$ and the real view. Therefore, we argue that the proposed scheme is secure. □

**Trapdoor unlinkability.** The proposed scheme will first expand the keyword vector $\overrightarrow{Q_W}$ into a vector $\overrightarrow{Q_{WE}}$ that contains "noise" in the process of generating trapdoors. Here, "noisy" refers to the random integer that is added in the third step of the *TrapdoorGen* algorithm when expanding $\overrightarrow{Q_W}$ into $\overrightarrow{Q_{WE}}$. Since the added "noise" is random, even the same $\overrightarrow{Q_W}$ will be expanded into a different $\overrightarrow{Q_{WE}}$. In addition, the $\overrightarrow{Q_{WE}}$ is randomly partitioned in the process of encrypting $\overrightarrow{Q_{WE}}$ via the secure KNN scheme. Based on the above two operations, the SSE-RK scheme can encrypt the same $Q$ into different trapdoors, thus achieving the unlinkability of the trapdoors.

**Keyword privacy.** Since the attacker can obtain the statistical information of the dataset under the known background model, it can make use of the word frequency information to analyze the query keywords in the trapdoor. To avoid such an attack, the proposed scheme expands the keyword vector $\overrightarrow{u_W}$ for each node when building the index tree. Specifically, vector $\overrightarrow{u_W}$ is extended with $L$ dimensions, where each dimension is set to a random "noise" $\epsilon_i$. In this way, the similarity score of any query is obfuscated by the value of $\sum \epsilon_i$. As $L$ increases, the probability of obtaining the same similarity score for the same query will be further reduced due to the interference of $\sum \epsilon_i$. Although the addition of "noise" increases privacy, it leads to a decrease in search precision. To balance privacy and precision, as analyzed in [21], we can make a trade-off by adjusting $L$ and $\epsilon_i$.

## 5. Performance Evaluation

In order to better demonstrate the performance of the proposed scheme, we will perform a theoretical analysis of the proposed scheme based on the experimental results. The experimental data are obtained from 50,000 documents, which were randomly selected from a real medical dataset named "OHSUMED" [39]. The experimental $PC$ contained an Intel(R) Core(TM) i7@2.90GHz CPU and 16 GB RAM. To better illustrate the merit of the proposed scheme, we performed simulation experiments on two schemes related to the proposed scheme, and we then compared them with our scheme. The performance comparison mainly focused on three aspects: index building, trapdoor generation, and ciphertext retrieval. For convenience, we denote the two schemes [8,9] to be compared by Miao18 and Wang19, and we listed some of the parameters that may affect the efficiency of these schemes in Table 2. In addition, we constructed Table 3 to show the efficiency comparison between these schemes. In the next sub-section, we will verify the theoretical analysis through experimental data and validate the effectiveness of the proposed scheme.

**Table 2.** Notations for the comparison analysis.

| | |
|---|---|
| $N$ | The number of keywords in the dictionary. |
| $d$ | The number of documents in the corpus. |
| $m$ | The dimension of the point in the document. |
| $L$ | The average length of a query range. |
| $M$ | The dimension of bloom filter used in Wang19. |
| $\theta$ | The average number of documents that match the query. |

**Table 3.** Comparative analysis of the scheme efficiency.

| Schemes | Index Building | Trapdoor Generation | Search |
|---|---|---|---|
| Miao18 | $2d(N+3m)^2$ | $(N+3m)^2$ | $d(N+3m)$ |
| Wang19 | $dLM^2$ | $LM^2$ | $L(M+m)$ |
| SSE-RK | $dm^2+dN^2$ | $(N+3m)^2+(4m)^2$ | $\theta(N+7m)$ |

### *5.1. Efficiency of Index Building*

According to Figure 6, the time cost of the index building in Miao18 is squarely related to $N$ and linearly associated with $d$. This is because the vector length of each document of Miao18 is $N+3m$. Thus, its index encryption process needs to perform $d(N+3m)^2$ product operations. For Wang19, since each leaf node contains $m$ points and a BF vector of length $M$, the $d$ leaf nodes in its index tree need to perform the $d(m+M)$ encryption operations of SHVE. In addition, since the internal node contains $m$ ranges $[a_i, b_i]$ and a BF vector, the internal node needs to perform the $dL(m+M)$ encryption operations of SHVE, where $L$ is the average length of $[a_i, b_i]$ and $i \in [1, m]$. Because the range of the internal node needs to include the points associated with all its leaf nodes, $L$ will increase as $d$ increases. Based on the above analysis, it can be inferred that the time consumption of index building in Wang19 is proportional to $d^2$. This conclusion is corroborated by the experimental results shown in Figure 6a. In addition, since Wang19 uses BF to index its keyword domain, the index-building time of Wang19 is independent of $N$. For the proposed scheme, since each leaf node of the index tree corresponds to a vector of length $N+3m$, the encryption of the leaf nodes needs to perform $d(N+3m)^2$ product operations. Moreover, the internal node corresponds to a vector of length $4m$, so the internal node needs to perform $d*m^2$ product operations. Since $m$ is much smaller than $d$ and $N$, we reckon that the time cost of index building in the SSE-RK scheme is squarely related to $N$ and linearly associated with $d$. The experimental results in Figure 6 are consistent with the theoretical analysis.

To sum up, as shown in Figure 6, the time cost of index building in Wang19 is much longer than that of Miao18 and the proposed scheme because its encryption process requires enumerating all points in the range associated with the internal node, and the SHVE encryption operation it uses is more time-consuming than the product operation. In addition, the time cost of index building in the SSE-RK scheme is slightly more than that of Miao18 because the proposed scheme needs to encrypt $d$ internal nodes.

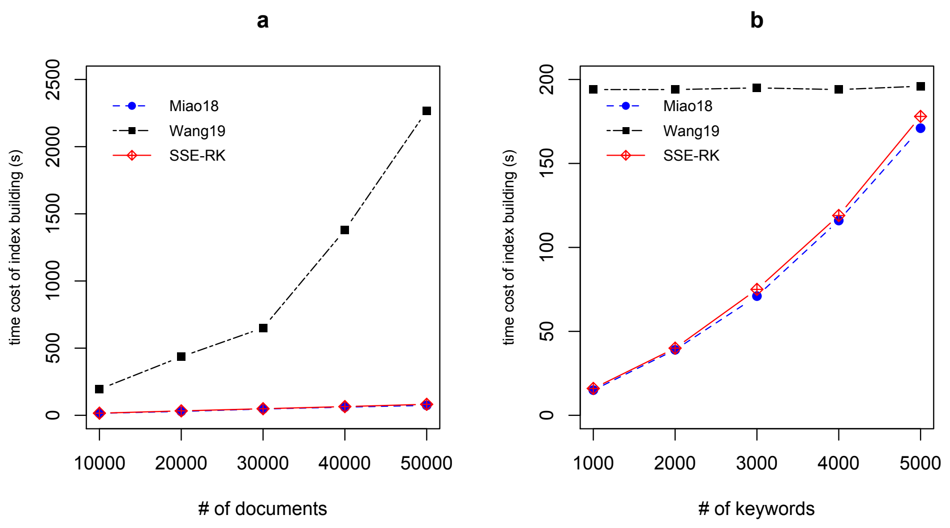


**Figure 6.** Impact of d (**a**) and N (**b**) on the time consumption of index building. N = (1000; 2000; 3000; 4000; 5000) and d = (10,000; 20,000; 30,000; 40,000; 50,000).

## 5.2. *Efficiency of Trapdoor Generation*

For Miao18, the vector length of its trapdoor is $N + 3m$; as such, its trapdoor generation process needs to execute $(N + 3m)^2$ product operations. For Wang19, assuming that its query range length is $L$, its trapdoor generation needs to execute the $L(m + M)$ key generation operations of SHVE since its query contains $L(m + M)$ vectors. The trapdoor of the proposed scheme contains two vectors. One's length is $N + 3m$ and the other's length is $4m$. Thus, its trapdoor generation process needs to execute $(N + 3m)^2 + (4m)^2$ product operations.

According to Figure 7a, it can be seen that all schemes are independent of $d$. Furthermore, the proposed scheme requires slightly more trapdoor generation time compared to Miao18. And the time cost of trapdoor generation in Wang19 is more than that of the other two schemes. This result could be explained by the fact that the time consumption of the key generation algorithm of SHVE used in Wang19 is more than that of the product operation in Miao18 and our scheme. In addition, according to Figure 7b, it can be seen that both Miao18 and the proposed scheme are squared with $N$, while Wang19 is independent of $N$. As $N$ keeps increasing, the time cost of trapdoor generation in Miao18 and the SSE-RK scheme will be higher than that of Wang19.

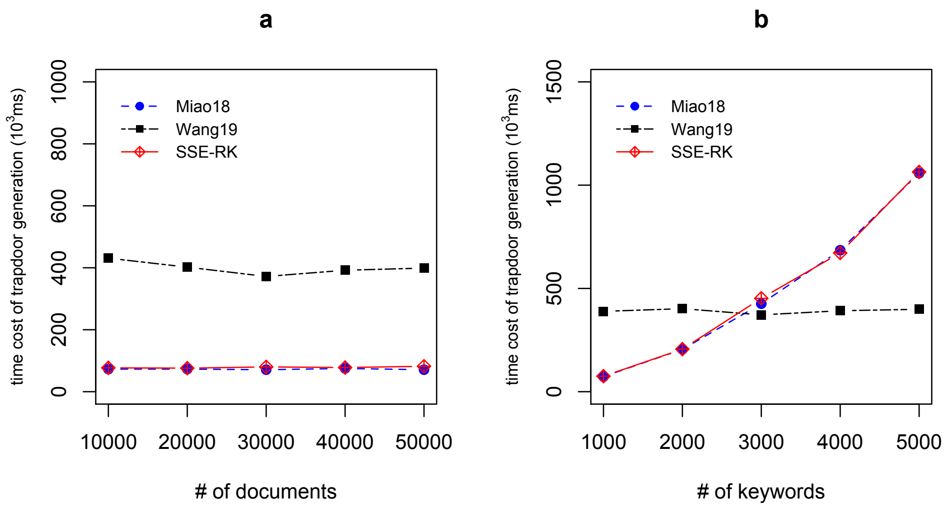


**Figure 7.** Impact of d (**a**) and N (**b**) on the time cost of trapdoor generation. N = (1000; 2000; 3000; 4000; 5000) and d = (10,000; 20,000; 30,000; 40,000; 50,000)

### *5.3. Efficiency of Search*

For Miao18, since the vector length of each document is $N + 3m$, the test algorithm requires performing $d(N + 3m)$ product operations. Considering that the index structure of Wang19 is a tree, assuming that $\theta$ represents the average number of documents that satisfy the query, its test algorithm needs to visit at least $\theta$ internal, as well as $\theta$ leaf, nodes. The test algorithm requires performing the $L(m + N)$ and $m + M$ decryption operations of SHVE for each internal node and leaf node, respectively. Thus, the total test time complexity is $\theta L(m + N)$. For the proposed scheme, like Wang19, it is necessary to visit at least $\theta$ internal, as well as $\theta$ leaf, nodes. For each internal node and leaf node, $4m$ and $3m + N$ product operations need to be performed, respectively. Thus, at least $\theta(7m + N)$ product operations need to be performed in total.

The results in Figure 8 corroborate the above analysis. As shown in Figure 8a, the proposed scheme has a sub-linear relationship with $d$ as $d$ increases. Compared with Miao18, the test time of the proposed scheme is lower since the proposed scheme utilizes the tree structure to reduce the access of a large number of irrelevant documents, thus making $\theta$ much smaller than $d$. Although both Wang19 and our scheme utilize the tree structure to improve the query efficiency, the search efficiency of the SSE-RK scheme is preferable to that of Wang19 since the decryption operation of SHVE is more time-consuming than the product operation. According to Figure 8b, the query time of Wang19 is independent of $N$, while that of Miao18 and our scheme grows slightly as N increases. Since the decryption operation of SHVE used by Wang19 is more time-consuming than the product operation, its query efficiency is still the lowest.

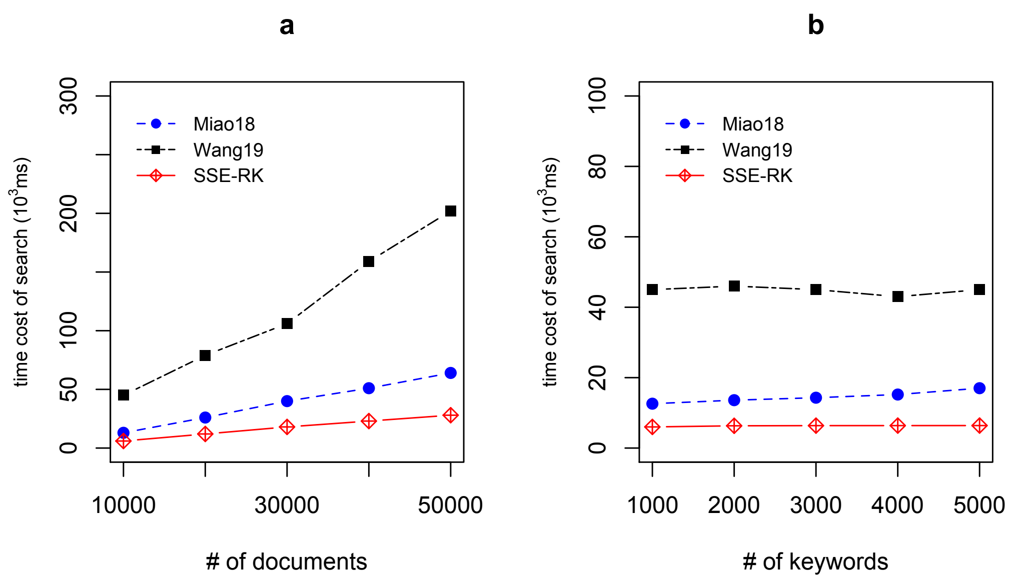


**Figure 8.** Impact of d (**a**) and N (**b**) on the time cost of search. N = (1000; 2000; 3000; 4000; 5000) and d = (10,000; 20,000; 30,000; 40,000; 50,000)

## *5.4. Discussion*

As shown in the experimental results, when d = 10,000 and $N = 1000$, the time cost of search in SSE-RK is 5.9 s while that in Miao18 is 12.7 s. However, the index building time of SSE-RK is nearly 2 s longer than that of Miao18. Since search operations are more frequent than index building operations, we can argue that SSE-RK is more practical than Miao18. Compared with Wang19, the proposed scheme has a significant improvement in index construction and query efficiency. Although the time cost of trapdoor generation in Wang19 is less than that of the proposed scheme when $N$ gradually increases, we reckon that the SSE-RK scheme is still practical since the trapdoor generation operation is a relatively small part of the overall user query process. The experimental analysis shows that Wang19 is less efficient. An objective reason for this is that Wang19 is designed to realize queries for arbitrary geometric ranges. In contrast, the proposed scheme and Miao18 are specifically designed to support range and keyword searches. In conclusion, based on the experimental results, we can find that the proposed scheme improves the query efficiency without sacrificing too much index building time. Considering the need for frequent queries on EMR data, we believe the proposed solution is more suitable for medical information systems.

Most of the existing SSE schemes only support keyword search. Compared to these schemes, the proposed scheme can support more complex query conditions. In order to quantify the cost of enhanced query functionality, we chose a highly efficient SSE scheme that only supports keyword search for an experimental comparison. We denote this scheme Zhang22 [38], and we show the experimental results in Figure 9. The experimental data show that the proposed scheme has an advantage in terms of index building time. This is because the internal nodes of the index tree of SSE-RK are constructed using range data, while that of Zhang22 are created using keyword vectors. Thus, the time complexity of the index tree building of SSE-RK is $dN^2 + dm^2$, while that of Zhang22 is $2dN^2$. The trapdoor generation time for the proposed scheme is a little higher than the one for Zhang22. This is because the trapdoor for SSE-RK will include range information in addition to keyword information. The search time for the proposed scheme is somewhat higher than the search time for Zhang22. This is because the query process of SSE-RK not only involves the similarity calculation of the keyword vectors, but also the determination of whether the ranges overlap. Zhang22, on the contrary, only needs to perform the similarity calculation of the keyword vectors. Therefore, the search time complexity of the proposed scheme is $7m + N$, while that of Zhang22 is $N$. The experiment results illustrate that the proposed scheme does incur a certain query cost when implementing more complex query conditions. Therefore, meaningful future work could be to improve the search efficiency of the scheme.

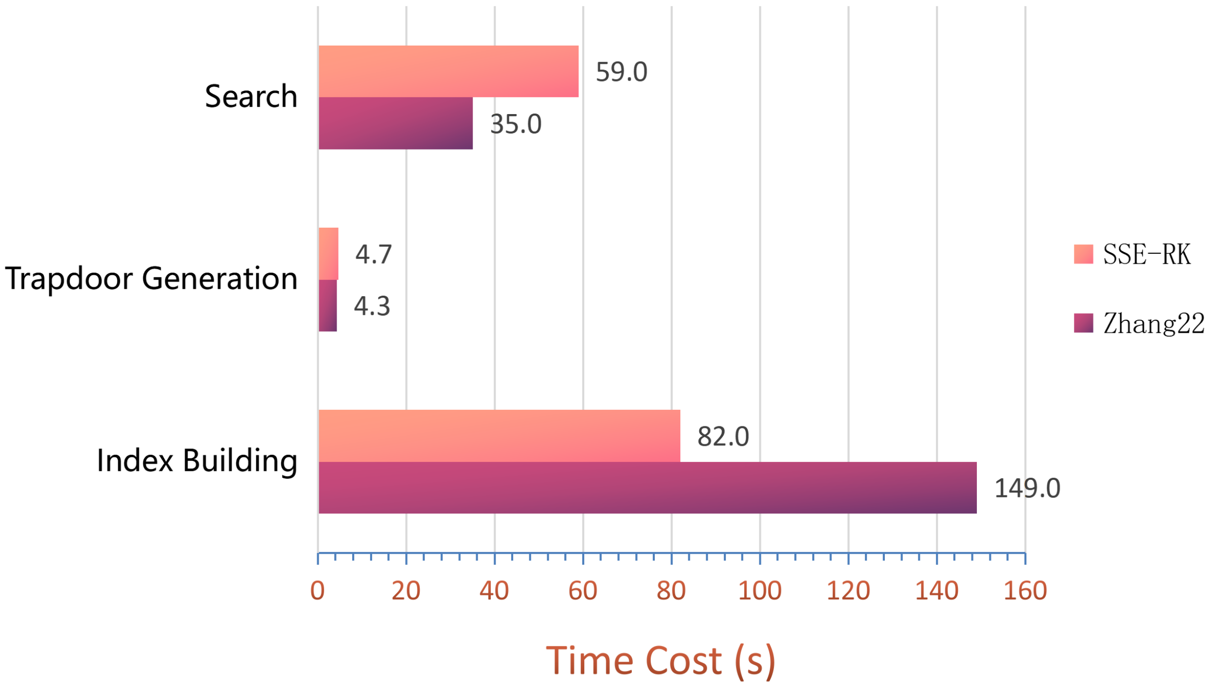


**Figure 9.** Comparison with the SSE scheme that supports only keyword search.

In addition, since the SSE-RK scheme can support both range and keyword search, it can also be utilized in applications such as location-based services [40] and protein prediction systems [41], etc. It is not difficult to find that the data in these applications will generally contain both numeric and textual types.

## 6. Conclusions

In this paper, we constructed a searchable encryption scheme that supports both range and keyword queries, and this scheme can perform a secure and fast search over encrypted EMR data. The construction of the SSE-RK scheme is divided into three main parts. Firstly, a keyword conversion method and two range conversion methods were proposed. These methods can transform keyword sets, range sets, and multi-dimensional points into vectors. Secondly, we designed an index tree-building algorithm. This algorithm makes use of the converted vectors to build all of the documents into a binary balanced tree in a bottom-up manner. Finally, the security of the SSE-RK scheme is ensured by encrypting each node in the index tree with a secure KNN algorithm. Furthermore, it is experimentally demonstrated that the query efficiency of the proposed scheme is sub-linearly related to the number of EMRs, and that it has better practicality than previous similar schemes.

In medical information systems, in addition to the range and keyword queries, user queries usually include more query conditions, such as in fuzzy queries, semantic queries, and Boolean queries. Therefore, our future work is to build a searchable encryption scheme that supports complex query conditions to enable a more accurate search over EMR data.

**Author Contributions:** Conceptualization, X.Y., Y.Z., and Y.L.; Data curation, Y.Z. and Y.L.; Formal analysis, X.Y., Y.Z., and Y.L.; Funding acquisition, Y.Z. and Y.L.; Methodology, X.Y. and Y.Z.; Software, Y.Z. and Y.W.; Validation, X.Y., Y.Z., and Y.L.; Writing—original draft, X.Y. and Y.Z.; Writing—review & editing, Y.Z., Y.L., and Y.W. 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 nos. 61972090 and 31872704), the Natural Science Foundation of Henan (grant no. 202300410339), and the Nanhu Scholars Program for Young Scholars of XYNU.

**Data Availability Statement:** The data used to support the findings of this study are available from the website, URL: https://trec.nist.gov/data/t9_filtering.html (accessed on 29 September 2023).

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

## References

1. Li, H.; Yang, Y.; Luan, T.H.; Liang, X.; Zhou, L.; Shen, X.S. Enabling fine-grained multi-keyword search supporting classified sub-dictionaries over encrypted cloud data. *IEEE Trans. Dependable Secur. Comput.* **2015**, *13*, 312–325. [CrossRef]
2. 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.
3. Miao, Y.; Weng, J.; Liu, X.; Choo, Ki.R.; Liu, Z.; Li, H. Enabling verifiable multiple keywords search over encrypted cloud data. *Inf. Sci.* **2018**, *465*, 21–37. [CrossRef]
4. Zhu, H.; Lu, R.; Huang, C.; Chen, L.; Li, H. An efficient privacy-preserving location-based services query scheme in outsourced cloud. *IEEE Trans. Veh. Technol.* **2015**, *65*, 7729–7739. [CrossRef]
5. Wang, B.; Li, M.; Wang, H. Geometric range search on encrypted spatial data. *IEEE Trans. Inf. Forensics Secur.* **2015**, *11*, 704–719. [CrossRef]
6. 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]
7. Tang, Q. Public key encryption schemes supporting equality test with authorisation of different granularity. *Int. J. Appl. Cryptogr.* **2012**, *2*, 304–321. [CrossRef]
8. Miao, Y.; Liu, X.; Deng, R.H.; Wu, H.; Li, H.; Li, J.; Wu, D. Hybrid keyword-field search with efficient key management for industrial internet of things. *IEEE Trans. Ind. Inform.* **2018**, *15*, 3206–3217. [CrossRef]
9. 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.
10. Lai, S.; Patranabis, S.; Sakzad, A.; Liu, J.K.; Mukhopadhyay, D.; Steinfeld, R.; Sun, S.; Liu, D.; Zuo, C. Result pattern hiding searchable encryption for conjunctive queries. In Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, Toronto, ON, Canada, 15–19 October 2018; pp. 745–762.
11. 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.
12. 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.
13. Goh, E.J. Secure indexes. *IACR Cryptol. Eprint Arch.* **2003**, *2003*, 216.
14. Byun, J.W.; Lee, D.H.; Lim, J. Efficient conjunctive keyword search on encrypted data storage system. In Proceedings of the European Public Key Infrastructure Workshop, Turin, Italy, 19–20 June 2006; Springer: Berlin/Heidelberg, Germany, 2006; pp. 184–196.
15. Ballard, L.; Kamara, S.; Monrose, F. Achieving efficient conjunctive keyword searches over encrypted data. In Proceedings of the International Conference on Information and Communications Security, Beijing, China, 10–13 December 2005; Springer: Berlin/Heidelberg, Germany, 2005; pp. 414–426.
16. Zhu, Y.; Ma, D.; Wang, S. Secure data retrieval of outsourced data with complex query support. In Proceedings of the 2012 32nd International Conference on Distributed Computing Systems Workshops, Macau, China, 18–21 June 2012; pp. 481–490.
17. Curtmola, R.; Garay, J.; Kamara, S.; Ostrovsky, R. Searchable symmetric encryption: Improved definitions and efficient constructions. *J. Comput. Secur.* **2011**, *19*, 895–934. [CrossRef]
18. Zerr, S.; Olmedilla, D.; Nejdl, W.; Siberski, W. Zerber$^{+R}$: Top-k retrieval from a confidential index. In Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology, Saint Petersburg, Russia, 24–26 March 2009; pp. 439–449.
19. 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]
20. 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]
21. 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]
22. Guo, C.; Zhuang, R.; Chang, C.; Yuan, Q. Dynamic multi-keyword ranked search based on bloom filter over encrypted cloud data. *IEEE Access* **2019**, *7*, 35826–35837. [CrossRef]

23. Liu, Z.; Wu, L.; Meng, W.; Wang, H.; Wang, W. Accurate range query with privacy preservation for outsourced location-based service in IOT. *IEEE Internet Things J.* **2021**, *8*, 14322–14337. [CrossRef]
24. Molla, E.; Rizomiliotis, P.; Gritzalis, S. Efficient searchable symmetric encryption supporting range queries. *Int. J. Inf. Secur.* **2023**, *22*, 785–798. [CrossRef]
25. Zheng, Y.; Lu, R.; Guan, Y.; Shao, J.; Zhu, H. Achieving efficient and privacy-preserving exact set similarity search over encrypted data. *IEEE Trans. Dependable Secur. Comput.* **2020**, *19*, 1090–1103. [CrossRef]
26. Gupta, B.B.; Lytras, M.D. Fog-enabled secure and efficient fine-grained searchable data sharing and management scheme for IoT-based healthcare systems. *IEEE Trans. Eng. Manag.* **2022**. [CrossRef]
27. 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]
28. Zhang, Y.; Li, Y.; Wang, Y. Efficient Searchable Symmetric Encryption Supporting Dynamic Multikeyword Ranked Search. *Secur. Commun. Netw.* **2020**, *2020*, 7298518. [CrossRef]
29. 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]
30. 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.
31. Boneh, D.; Crescenzo, G.D.; 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; Springer: Berlin/Heidelberg, Germany, 2004; pp. 506–522.
32. 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, Republic of Korea, 23–25 August 2004; Springer: Berlin/Heidelberg, Germany, 2004; pp. 73–86.
33. 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; Springer: Berlin/Heidelberg, Germany, 2008; pp. 146–162.
34. 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*, 5280806. [CrossRef]
35. Kim, I.; Hwang, S.O.; Park, J.H.; Park, C. An Efficient Predicate Encryption with Constant Pairing Computations and Minimum Costs. *IEEE Trans. Comput.* **2016**, *65*, 2947–2958. [CrossRef]
36. Zhang, R.; Wang, J.; Song, Z.; Wang, X. An enhanced searchable encryption scheme for secure data outsourcing. *Sci. China Inf. Sci.* **2020**, *63*, 132102. [CrossRef]
37. 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.* **2019**, *18*, 1080–1094. [CrossRef]
38. He, W.; Zhang, Y.; Li, Y. Fast, Searchable, Symmetric Encryption Scheme Supporting Ranked Search. *Symmetry* **2022**, *14*, 1029. [CrossRef]
39. Hersh, W.; Buckley, C.; Leone, T.J.; Hickam, D. OHSUMED:An interactive retrieval evaluation and new large test collection for research. In Proceedings of the 17th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, Dublin, Ireland, 3–6 July 1994; pp. 192–201.
40. Shin, K.G.; Ju, X.; Chen, Z.; Hu, X. Privacy protection for users of location-based services. *IEEE Wirel. Commun.* **2012**, *19*, 30–39. [CrossRef]
41. 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.