Multi-Keyword Ranked Searchable Encryption with the Wildcard Keyword for Data Sharing in Cloud Computing

Jinlu Liu[1], Bo Zhao[1], Jing Qin[1,2], Xi Zhang[1] and Jixin Ma[3]

[1]*School of Mathematics, Shandong University, Jinan, 250100 Shandong, China*
[2]*State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Sciences, Beijing 100093, China*
[3]*School of Computing and Mathematical Sciences University of Greenwich, London,UK*
*****Corresponding author: qinjing@sdu.edu.cn*

Multi-keyword ranked searchable encryption (MRSE) supports multi-keyword contained in one query and returns the top-k search results related to the query keyword set. It realized effective search on encrypted data. Most previous works about MRSE can only make the complete keyword search and rank on the server-side. However, with more practice, users may not be able to express some keywords completely when searching. Server-side ranking increases the possibilities of the server inferring some keywords queried, leading to the leakage of the user's sensitive information. In this paper, we propose a new MRSE system named 'multi-keyword ranked searchable encryption with the wildcard keyword (MRSW)'. It allows the query keyword set to contain a wildcard keyword by using Bloom filter (BF). Using hierarchical clustering algorithm, a clustering Bloom filter tree (CBF-Tree) is constructed, which improves the efficiency of wildcard search. By constructing a modified inverted index (MII) table on the basis of the term frequency-inverse document frequency (TF-IDF) rule, the ranking function of MRSW is performed by the user. MRSW is proved secure under adaptive chosen-keyword attack (CKA2) model, and experiments on a real data set from the web of science indicate that MRSW is efficient and practical.

Keywords: searchable encryption; bloom filter; hierarchical clustering; TF-IDF; multi-keyword ranked; wildcard keyword

1. INTRODUCTION

Symmetric searchable encryption (SSE) is a cryptographic primitive that addresses the search over encrypted data. It allows a data owner to encrypt documents using a special encryption algorithm and outsource them to an external server so that other authorized users (including the data owner) can generate the trapdoor corresponding to the query keyword. Given a trapdoor, the server searches for encrypted documents and returns the search results to the user.

SSE was first introduced by Song *et al.* [1] in 2000. Their scheme adopts a method similar to stream ciphers for encryption and needs to compare the ciphertext word by word to find the searched keywords. Goh [2] presented an index-based scheme to improve search efficiency. Subsequently, a series of SSE schemes have been proposed [3, 4]. All these schemes can search for documents containing a specific single keyword securely and efficiently.

To enhance the searching experience of users, Wang et al. [5] firstly defined and solved the problem of multi-keyword ranked search over encrypted data (MRSE). In query phase, users can generate the trapdoor corresponding to multi-keyword, which improves the search accuracy. In search phase, the search results are to be ranked, and the server returns the top-k documents that are most relevant to query keywords. This makes users find the most relevant one without decrypting all the documents and decreases the unnecessary communication traffic.

While providing efficient multi-keyword ranked search, almost all of existing MRSE schemes [5–13] can only search the complete keyword. It is common that users are unable

to accurately and completely express one of the keywords when performing a multi-keyword ranked search. For example, in the cloud medical system, hospitals build medical record indexes by extracting the specific date of the patient's medical record creation (such as 5 April 2021), the basic information of patients, the disease name and other keywords, and stores indexes and encrypted medical records in the cloud server. In order to conduct collaborative diagnosis and treatment, hospital A needs to share the medical record resources of diabetic patients generated in May 2021 with hospital B. We can extract the keywords 'diabetic' and 'May 2021'. Nevertheless, the 'May 2021' cannot be expressed in the form of a pre-set keyword. Using the existing MRSE system, hospital B needs to enumerate all possible keywords such as '2021/05/01', '2021/05/02',..., '2021/05/31', and then perform searches for all cases. This is not an efficient way.

For the ranking operation, in most previous MRSE systems, it is executed by the server. However, for the cloud server with a large amount of background knowledge (for example, the correlation between given trapdoors, relevant data set statistics information, etc.), it can combine this knowledge with the ranking results to perform statistical attacks to infer some keywords queried. These keywords usually involve some sensitive information about the user. For instance, in cloud medical system, keywords are often related to the patient's illness. To protect data privacy, users will prefer to rank locally to reduce the possibilities of statistical attacks of the server.

1.1. Contribution

In this paper, we propose a multi-keyword ranked searchable encryption with the wildcard keyword (MRSW) that enables users to rank the candidate documents and then request the server to return the top-k most relevant documents in search phase. With MRSW, hospital B can generate trapdoor of keywords '2021/05/??' and 'diabetic' for searching. Specifically, there are three main techniques in designing our scheme. (1) Clustering the keywords in the keyword dictionary by using a hierarchical clustering algorithm. The keyword similarity is measured by the Hamming distance between the Bloom filters (BF) corresponding to keywords and the high similar keywords are in one cluster. Based on the clustering result, the document clustering bit vector (DCB-vector) and query clustering bit vector (QCB-vector) are generated. By calculating the inner product of these two vectors, we can filter out the documents that are entirely impossible to match with the query keywords and obtain the candidate document identifier (CID) set. (2) Based on the keyword clustering, a clustering Bloom filter tree (CBF-Tree) can be generated. Each leaf node represents a cluster of keywords, which allows us to quickly filter out the tree branches that do not match with the wildcard keyword and find the matching keywords. (3) To make the ranking function performed by the user, a modified inverted index table (MII) is constructed based on the TF-IDF rule and pseudo-random function (PRF). Using the MII, the cloud server calculates the encrypted relevance scores of the documents in the CID and sends these encrypted relevance scores and their corresponding identifiers to the user. The user obtains the plaintext scores and ranking results by decrypting the ciphertext scores.

The primary contributions are summarized below:

1. We design a novel MRSW scheme. It supports a more flexible search function, namely, multi-keyword ranked search where the search keyword set contains a wildcard keyword.
2. According to the clustering of keywords, a CBF-Tree is constructed, which improves the efficiency of the wildcard search.
3. In contrast to the previous MRSE ranked on the server-side, our scheme realizes ranking on the user-side through simple addition and subtraction. It reduces the chance of statistical attacks effectively.

1.2. Related Works

1.2.1. Multi-Keyword Ranked Searchable Encryption

The first study of MRSE was put forward by Cao et al. in 2011 [5]. They used a novel index structure. Concretely, the documents and query keywords are expressed as 0/1 bit-vectors with a dimension equal to the number of keywords. Using secure k-nearest neighbor (KNN) encryption and 'coordinate matching', documents can be ranked by the number of keywords they contain. The frequency of keywords reflects the importance of keyword to the document to a certain extent, but [5] did not consider this. Therefore, in 2013, Cao et al. [6] optimized the representation method of document vectors. Aimed at the importance of keyword frequency to documents, the TF-IDF rule and vector space model (VSM) were introduced to improve search accuracy.

The indexes of [5] and [6] are linear. To achieve better search efficiency than linear ones, follow-up researchers have proposed many novel tree index structures. For example, in 2015, Chen et al. [8] proposed the hierarchical clustering index tree; in 2015, Xia et al. [9] introduced the keyword balanced binary index tree; in 2019, Guo et al. [10] constructed an index tree by utilizing BF.

In the schemes mentioned above, the ranking functions are all performed by the cloud server, but server-side ranking inevitably leads to statistical information leakage. Therefore, Yu et al. [7] introduced a two-round interactive MRSE by utilizing VSM and homomorphic encryption (HE), in which the ranking function is performed by the user.

Apart from the exact search, many fuzzy MRSE schemes also have been proposed. Wang et al. [11] proposed a secure fuzzy MRSE under known ciphertext model by extracting the bigram characters of keywords and using local sensitive hashing (LSH) technology. Furthermore, Fu et al. [12] extracted uni-gram characters of keywords to improve search accuracy.

TABLE 1. Functionality Comparise.

Scheme	Key	Search Functionality			
		single wildcard	multi-keyword rank	one wildcard keyword contained in query keyword set	rank performer
[14]	Symmetric	✓	×	⊥	⊥
[15]	Asymmetric	✓	×	⊥	⊥
[6]	Symmetric	×	✓	×	Server
[7]	Asymmetric	×	✓	×	User
Ours	Symmetric	✓	✓	✓	User

In 2019, Cao et al. [13] pointed out that [12] could not recognize the anagrams. To solve this problem, a novel method for converting keywords into 'order-preserved uni-gram (OPU)' vectors was proposed in [13].

1.2.2. Wildcard Searchable Encryption

Generally, there are mainly two types of wildcard: '?' and '*'. '?' can represent one character, which is called a single-character wildcard, and '*' can represent any number of characters, which is called a multi-character wildcard. In 2010, Sedghi et al. [16] presented a method to convert hidden vector encryption (HVE) into public key searchable encryption (PKSE) that supports wildcard search and proposed a specific HVE scheme. Their scheme is not efficient because many modular exponential operations are needed in encryption, trapdoor generation and search algorithms and several bilinear pairing operations are required in search algorithm.

Bösch et al. [17] presented a wildcard SSE in 2011. It is more efficient than [16] and each wildcard can represent multiple characters. However, it just enumerates all the wildcardified variants of the keyword and inserts these keywords into a BF, which leads to high preprocessing costs and large storage capacity.

In 2012, Suga et al. [18] presented an SSE scheme that supports multiple types of searches. Specifically, a BF is constructed for each keyword by extracting the keyword's characteristics. That is to say, the index is an inverted index. Although this scheme does not need to enumerate the wildcardified variants of keyword, due to the limitation of the characteristic extraction method, a wildcard can only represent one character, which is not practical. For this limitation, in 2016, Hu et al. [14] presented a new keyword characteristic extraction method so that each wildcard can represent any number of characters. In 2016, Zhao et al. [19] introduced an SSE scheme that can support both single-character wildcard '?' and multi-character wildcard '*' by improving the keyword characteristic extraction method in Suga's scheme.

An obvious drawback of BF-based searchable encryption schemes [14, 17–19] is the inevitability of false positive. Therefore, to eliminate the drawback caused by BF, Yang et al. [15] constructed a flexible wildcard searchable encryption scheme using HE. The query keyword can contain zero to two wildcards. Wildcards can appear anywhere in the keyword and a wildcard can replace any number of characters. However, when there are multiple wildcards in the query keyword, it will result in multi-level loop nesting. So this scheme is not applicable to the search for query keyword with three or more wildcards.

In Table 1, we give the functionality comparisons for our MRSW scheme with existing works [14], [15], [6], [7].

As we have seen, many researchers have focused on multi-keyword ranked and wildcard searchable encryption. However, almost all systems do not support the situation where a single query contains multiple keywords and one of the keywords is the wildcard keyword. Besides, the ranking functions are performed on the server-side in most studies about MRSE. Therefore, it is necessary to propose the MRSW.

1.3. Organization

The rest of this paper is organized as follows: Section 2 gives some notations and preliminaries. Section 3 presents the system and security model of MRSW. Section 4 describes the concrete algorithms of MRSW. Security and performance analysis of MRSW are showed in Section 5 and Section 6, respectively. Finally, the conclusion of this paper is in Section 7.

2. NOTATIONS AND PRELIMINARIES

2.1. Notations

The notations used in this paper are listed in Table 2.

2.2. Preliminaries

2.2.1. Bloom Filter

Bloom filter (BF) [20] is an effective data structure for judging whether an element belongs to a collection. It uses an array to represent a collection. Initially, each position of the array is set to 0. For a set S, calculate l hash values of each element in S and set the position corresponding to these values to 1. When judging whether an element q is in the set S, calculate l hash values of q and check the bits at these l positions, as long as there is bit 0, then q must not be in S; otherwise, $q \in S$ or yields

TABLE 2. Notations.

Notation	Meaning
D	plaintext document set, $D = \{d_1, d_2, ..., d_m\}$
ID	identifier set of document set D, $ID = \{id_1, id_2, ..., id_m\}$
C	encrypted document set, $C = \{c_1, c_2, ..., c_m\}$
W	keyword dictionary, $W = \{w_1, w_2, .., w_n\}$
CL	keyword clustering result list, $CL = \{C_1, C_2, ..., C_\tau\}$
VC_{d_i}	τ-dimensional DCB-vector of d_i
D_{w_i}	document set that containing w_i
Q	multi-keyword query, $Q = \{Q_e, q_w\}$, where $Q_e = \{q_1, q_2, .., q_l\}$ is the exact keyword set and q_w is the wildcard keyword
VC_{Q_e}	τ-dimensional QCB-vector of Q_e
CID	the set of candidate document identifiers for query Q_e

a false positive, this is because each position may have been set to 1 by some other elements other than q.

2.2.2. *Hamming Distance*

Hamming distance is defined as the number of different characters in the corresponding positions between two strings of equal length. In other words, the number of characters needed to be changed from one string to another, such as the hamming distance between 1011 100 and 1001 000 is 2. Therefore, hamming distance can measure the similarity between two strings of equal length.

2.2.3. *AGglomerative NESting*

Hierarchical clustering attempts to divide the data at different levels to form a tree-shaped cluster structure. The data collection can be divided by either a 'bottom-up' aggregation strategy or a 'top-down' split strategy.

In this paper, we use a 'bottom-up' aggregation strategy hierarchical clustering algorithm 'AGglomerative NESting' (AGNES). AGNES regards each sample in the data collection as an initial cluster, and then finds the closest two clusters and merges them in each step of the algorithm. This process is repeated until the preset number of clusters is reached. Generally, we can use three methods to calculate the distance between clusters. For example, given the clusters C_i and C_j, the following formulas can be used to calculate the distance:

$$\textit{Minimum Distance} : d_{min}(C_i, C_j) = \min_{x \in C_i, z \in C_j} dist(x, z) \quad (1)$$

$$\textit{Maximal Distance} : d_{max}(C_i, C_j) = \max_{x \in C_i, z \in C_j} dist(x, z) \quad (2)$$

$$\textit{Average Distance} : d_{avg}(C_i, C_j) = \frac{1}{|C_i||C_j|} \sum_{x \in C_i} \sum_{z \in C_j} dist(x, z) \quad (3)$$

.

Algorithm 1 AGNES

Input: Sample collection $D = \{x_1, x_2, .., x_m\}$;
Cluster measurement function d;
Number of clusters τ.
Output: Cluster partition $C = \{C_1, C_2, ..., C_\tau\}$

1: **for** $j = 1, 2, ..., m$ **do**
2: $C_j = \{x_j\}$
3: **end for**
4: **for** $i = 1, 2, ..., m$ **do**
5: **for** $j = 1, 2, ..., m$ **do**
6: $M(i, j) = d(C_i, C_j)$;
7: $M(j, i) = M(i, j)$
8: **end for**
9: **end for**
10: Set the current number of clusters: $q = m$
11: **while** $q > \tau$ **do**
12: Find the two closest clusters C_{i^*} and C_{j^*};
13: Merge C_{i^*} and C_{j^*}: $C_{i^*} = C_{i^*} \cup C_{j^*}$;
14: **for** $j = j^* + 1, j^* + 2, .., q$ **do**
15: Renumber the cluster C_j as C_{j-1}
16: **end for**
17: Delete the j^*-th row and j^*-th column of matrix M;
18: **for** $j = 1, 2, ..., q - 1$ **do**
19: $M(i^*, j) = d(C_{i^*}, C_j)$;
20: $M(j, i^*) = M(i^*, j)$
21: **end for**
22: $q = q - 1$
23: **end while**

The AGNES algorithm is described in Algorithm 1. In lines 1 to 9, the algorithm first initializes the initial cluster containing only one sample and the corresponding distance matrix; then in the 11th to 23rd lines, AGNES continuously merges the closest clusters and updates the distances of the merged clusters. The above process is repeated until the preset number of clusters is reached.

2.2.4. *TF-IDF*

Some multi-keyword SSE schemes only support boolean queries, that is, whether a document matches a query. In order to improve search accuracy and reduce bandwidth cost, it is necessary to rank the search results in the order of their relevance to query keywords.

A natural method to measure relevance is to calculate scores. We adopt the universally accepted TF-IDF rule for measurement. TF-IDF includes two attributes-*term frequency* and *inverse document frequency*. *Term frequency* ($tf_{t,f}$) refers to the number of times the term t appears in document f. *Document frequency* (df_t) refers to the number of documents containing term t and the *inverse document frequency* (idf_t) is defined as $idf_t = log\frac{N}{df_t}$, where N is the number of all

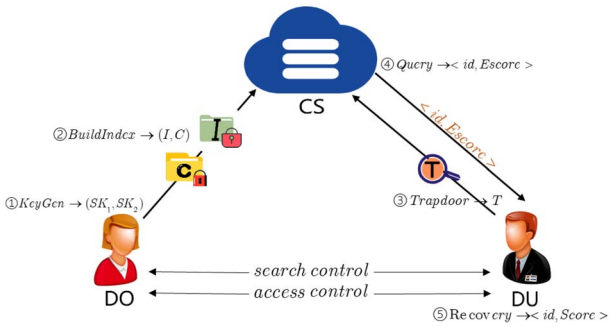


FIGURE 1. System Framework.

documents. A low-frequency term tends to have a higher idf_t, while a high-frequency term has a lower idf_t. The formula for calculating the relevance score is as follows: $tf\text{-}idf = tf_{t,f} \times idf_t$.

3. MRSW SYSTEM

3.1. System Framework

In MRSW, as shown in Fig. 1, there are three entities: Data Owner (DO), Data User (DU) and Cloud Server (CS). Both the DO and DU are trusted. The CS is considered as 'honest and curious'. In other words, the CS will strictly implement the protocol, and try to use its resources to obtain as much secret information as possible.

DO is responsible for processing the original data, generating secure index and encrypted document set; CS receives and stores the data from the DO, and performs the search operation according to the DU's request; DU is the user authorized by DO and can search the data of DO stored in the CS.

DEFINITION 3.1. *(MRSW System). The MRSW system consists of the following five algorithms.*

- $KeyGen(1^n) \rightarrow SK$. This is a probabilistic key generation algorithm run by the DO. It takes as input a security parameter n and outputs key SK.
- $BuildIndex(D, SK) \rightarrow (I, C)$. On input the key SK and document set D, the DO runs the algorithm to generate index I and encrypted document set C, and outsources them to the CS.
- $Trapdoor(Q, SK) \rightarrow T$. DU can use a multi-keyword query Q (which may contain a wildcard keyword) to retrieve documents of interest. For this purpose, the DU generates a search trapdoor T and submits it to the CS.
- $Query(T, I, C) \rightarrow \langle id, Escore \rangle$. This is a deterministic algorithm run by the CS. After receiving a trapdoor T, the CS performs the search on the index I and finally returns the identifier-encrypted relevance score pair $\langle id, Escore \rangle$ of each candidate document.
- $Recovery(\langle id, Escore \rangle, SK) \rightarrow \langle id, Score \rangle$. DU runs this algorithm to decrypt $Escore$ and get the identifier-relevance score pair $\langle id, Score \rangle$ of each candidate document. Furthermore, the CS can get the identifier of the top-k documents by ranking according to the relevance score.

3.2. Security Model

We extend the widely accepted simulated-based framework captured by real-world versus ideal-world formalization [4, 21] to the case of multi-keyword ranked search and then prove the security of MRSW. Formally, the security says that only by giving the output of the leakage function, the adversary's view can be simulated.

DEFINITION 3.2. *(Multi-keyword rank CKA2-Security). Let* $\Pi = (KeyGen, BuildIndex, Trapdoor, Query, Recovery)$ *be an MRSE scheme supporting a wildcard keyword contained in a query as in definition 1.1, and consider the following probabilistic experiments where* $\mathcal{A}$ *is an adversary,* $\mathcal{S}$ *is a simulator, and* $\mathcal{L}_1$ *and* $\mathcal{L}_2$ *are (stateful) leakage functions:*

$\mathbf{Real}^{\Pi}_{\mathcal{A}}(\mathbf{n})$: the challenger begins by running $KeyGen(1^n)$ to generate key SK. $\mathcal{A}$ outputs D and receives $BuildIndex(D, SK) \rightarrow (I, C)$ from the challenger. $\mathcal{A}$ makes a polynomial number of adaptive multi-keyword ranked queries and, for each query Q, receives a trapdoor $Trapdoor(Q, SK) \rightarrow T$ from the challenger. Finally, $\mathcal{A}$ returns a bit b that is output by the experiment.

$\mathbf{Ideal}^{\Pi}_{\mathcal{A},\mathcal{S}}(\mathbf{n})$: $\mathcal{A}$ outputs D. Given $\mathcal{L}_1(D)$, $\mathcal{S}$ generates and sends a pair (I, C) to $\mathcal{A}$. $\mathcal{A}$ makes a polynomial number of adaptive multi-keyword ranked queries and for each query Q, $\mathcal{S}$ is given $\mathcal{L}(D, Q)$ and returns a trapdoor T. Finally, $\mathcal{A}$ returns a bit b that is output by the experiment.

The Π is $(\mathcal{L}_1, \mathcal{L}_2)$-secure against adaptive chosen-keyword attacks (CKA2) if for all probability polynomial time (PPT) adversaries $\mathcal{A}$, there exists a PPT simulator $\mathcal{S}$ such that

$$|Pr[\mathbf{Real}^{\Pi}_{\mathcal{A}}(\mathbf{n}) = 1] - Pr[\mathbf{Ideal}^{\Pi}_{\mathcal{A},\mathcal{S}}(\mathbf{n}) = 1| \leq negl(n),$$

where $negl(n)$ is a negligible function in n.

4. MRSW SCHEME

4.1. $KeyGen \rightarrow SK$

The algorithm includes the following two sub algorithms.

- $\mathbf{KeyGen_1}(\mathbf{1^n}) \rightarrow \mathbf{SK_1}$. Given security parameter n, DO randomly chooses the following uniformly from the corresponding domains:

1. an *n-bit* string *sk* as a symmetric key for a pseudo-randomness against chosen plaintext attacks (PCPA)-security symmetric key encryption (SKE) scheme;
2. a pseudo random function $f : \{0,1\}^* \times \{0,1\}^n \rightarrow \{0,1\}^n$ and $r+2$ random keys with n-bits: $K = \{k_1, k_2, ..., k_r\}$, k_w and k_s.

The output of this algorithm is $SK_1 = \{sk, K = \{k_1, k_2, ..., k_r\}, k_w, k_s\}$.

- $\mathbf{KeyGen_2}(\tau) \rightarrow \mathbf{SK_2}$. When the clustering number τ of keywords is determined, DO randomly generates key set SK_2, including:

1. a τ-dimensional bit-vector S;
2. two $\tau \times \tau$ invertible matrices M_1, M_2.
Namely, $SK_2 = \{S, M_1, M_2\}$.

Therefore, $SK = (SK_1, SK_2)$.

4.2. *BuildIndex* $\rightarrow (I, C)$

The algorithm can be run in the following steps.

Before constructing index, the DO firstly clusters keywords.

- $\mathbf{GenBF}(\mathbf{W}, \mathbf{K}) \rightarrow \mathbf{BF}$. First, the keyword set $W = \{w_i\}_{1 \le i \le n}$ is extracted from document set $D = \{d_i\}_{1 \le i \le m}$, and then construct the corresponding BF for each $w_i \in W$ by extracting characteristics (We adopt the same keyword characteristic extraction method as [14]). The specific steps are as follows:

1. Initialize a BF for w_i;
2. Assume that keyword w_i includes h characters: $w_i[1]$,...,$w_i[h]$, we can obtain a collection: $S_{index} = \{w_i[1]||1, ..., w_i[h]||h, w_i[1]||(-h), .., w_i[h]||(-1), w_i[1]||0, ..., w_i[h]||0\}$, i.e. it contains the positive order, reverse order and existence of every character;
3. For each element s in S_{index}, calculate its corresponding PRF values $\{F(s, k_i)\}_{1 \le i \le r}$ and add the r values to the BF, namely the positions of the r values are set to 1.

- $\mathbf{GenClustering}(\mathbf{W}, \tau) \rightarrow \mathbf{CL}$. Firstly, regarding each BF of each keyword as a binary bit string and using the hamming distance between bit strings to measure the similarity between keywords. Then, we can adopt the AGNES algorithm to cluster keywords, where d_{avg} is used as the cluster measurement function and the number of clusters τ can be set according to our needs. Finally, we can obtain the clustering result list $CL = \{C_1, C_2, ..., C_\tau\}$.
We give a concrete example to describe the above algorithm, and we will still use this example when we need to describe other algorithms later.

Example1 : Fig. 2 illustrates an example of clustering keywords using the AGNES algorithm. We assume that D contains six documents, W concludes 10 keywords and the clusters number τ is set to 4.
Then, the DO generates index I and encrypted document set C.

- $\mathbf{GenDCBVector}(\mathbf{D}, \mathbf{CL}, \mathbf{SK_2}) \rightarrow \mathbf{I_1}$. According to CL, DO firstly generates a τ-dimensional Document Clustering Bit vector (DCB-vector) VC_{d_i} for each $d_i \in D$. Specifically, each component of the vector represents a cluster of keywords. If the document contains one or more keywords in the cluster, the corresponding component is 1; otherwise it is 0. The set of generated DCB-vectors is denoted as $VC_D = \{VC_{d_1}, VC_{d_2}, ..., VC_{d_m}\}$. Then DO uses secure KNN to encrypt every DCB-vector VC_{d_i}. Specifically, the encryption method is as follows: DO first generates two random vectors $\{VC_{d_i,1}, VC_{d_i,2}\}$ according to S. In detail, if $S[j] = 0$, $VC_{d_i,1}[j] = VC_{d_i,2}[j] = VC_{d_i}[j]$; if $S[j] = 1$, randomly generate $VC_{d_i,1}[j]$ and $VC_{d_i,2}[j]$ so that the sum of the two is $VC_{d_i}[j]$. Lastly, DO uses M_1 and M_2 to encrypt $VC_{d_i,1}$ and $VC_{d_i,2}$, respectively, to generate index $I_1 = \{M_1^T VC_{d_i,1}, M_2^T VC_{d_i,2}\}_{1 \le i \le m}$.
- $\mathbf{GenCBFTree}(\mathbf{BF}, \mathbf{F}, \mathbf{k_w}) \rightarrow \mathbf{I_2}$. According to the keyword clustering result, generate a Cluster Bloom Filter Tree (CBF-Tree). For simplicity of explanation, we assume that τ is a power of 2. Specifically, DO first uses the key k_w to calculate the PRF value $F(w, k_w)$ of each keyword in Q and uses this value as the unique identifier of the keyword. Then as shown in Fig. 3:

1. According to the clustering process, divide the τ keyword clusters into pairs of keyword clusters. Each leaf node represents a cluster of keyword clusters and stores the bitwise OR of BFs corresponding to all keywords in this cluster.
2. Each leaf node is directly associated with the identifier and BF of all keywords in this cluster.
3. Compute the bitwise OR of each pair of keyword cluster (BF) and assign the result (BF) to the parent node of the pair in the CBF-Tree.
4. Repeat step 3 for higher levels of the tree until the root node is reached. Note that the BF of the root node of CBF-Tree contains the BF corresponding to the entire keyword set.

We denote the CBF-Tree as index I_2.

- $\mathbf{GenMII}(\mathbf{D}, \mathbf{W}, \mathbf{F}, \mathbf{k_s}) \rightarrow \mathbf{I_3}$. In order to realize the ranking function of search result, as illustrated in Fig. 4, DO builds a table similar to the inverted index, which we call MII table, to record the encrypted relevance score between the document and the keyword. Concretely, the encrypted relevance scores are calculated as

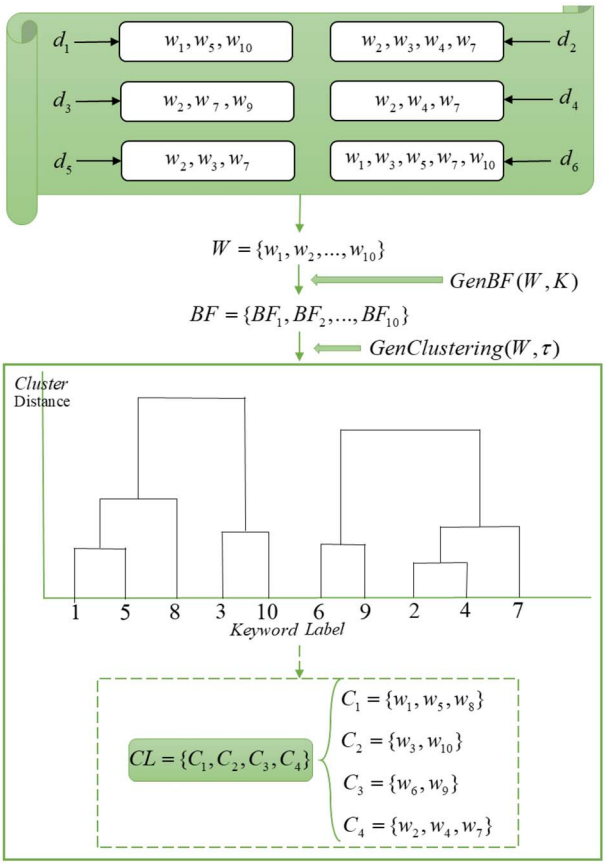


FIGURE 2. Example of GenClustering.

follows:

1. Calculate the plaintext relevance score $score_{i,j} (1 \leq i \leq n, 1 \leq j \leq m)$ between each keyword w_i and each document d_j according to TF-IDF rule;
2. Use PRF F to encrypt each $score_{i,j}$ into $Escore_{i,j}$:

$$Escore_{i,j} = score_{i,j} + F(i||id_j, k_s). \tag{4}$$

We denote the MII as index I_3.

- **Enc**$(\mathbf{D}, \mathbf{sk}) \rightarrow \mathbf{C}$. For $1 \leq i \leq m$, let $c_i \leftarrow SKE.Enc(sk, d_i)$ and $C = \{c_1, c_2, .., c_m\}$.
Finally, DO uploads $I = \{I_1, I_2, I_3\}$ and C to the CS.

4.3. *Trapdoor* $\rightarrow T$

The algorithm includes the following three sub algorithms.

- **GenQCBVector**$(\mathbf{Q_e}, \mathbf{CL}, \mathbf{SK_2}) \rightarrow \mathbf{T_1}$. According to CL, DU firstly generates a τ-dimensional Query Clustering Bit vector (QCB-vector) VC_{Q_e} for exact keyword set Q_e. Specifically, each component of the vector represents a cluster of keywords. If Q_e contains one or more keywords in the cluster, the corresponding component is 1, otherwise it is 0. Then use secure KNN to encrypt plaintext QCB-vector VC_{Q_e}. Similar to the encryption of DCB-vector, the QCB-vector is split into two random vector $\{VC_{Q_e,1}, VC_{Q_e,2}\}$. The difference is that if $S[j] = 1$, $VC_{Q_e,1}[j] = VC_{Q_e,2}[j] = VC_{Q_e}[j]$; otherwise, set $VC_{Q_e,1}[j]$ and $VC_{Q_e,2}[j]$ to two random values that add up to $VC_{Q_e}[j]$. Lastly, the trapdoor $T_1 = \left\{M_1^{-1} VC_{Q_e,1}, M_2^{-1} VC_{Q_e,2}\right\}$.

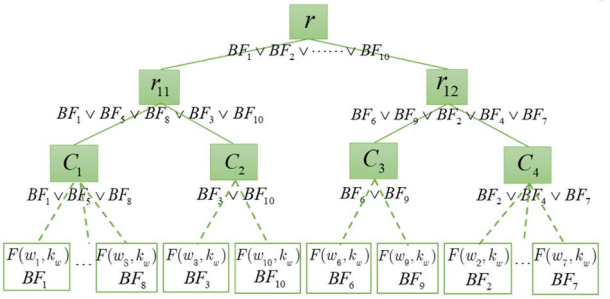


FIGURE 3. CBF-Tree.

Keyword ID \ Document ID	id_1	id_2	...	id_m
$F(w_1, k_w)$	$Escore_{1,1}$	$Escore_{1,2}$	⋯	$Escore_{1,m}$
$F(w_2, k_w)$	$Escore_{2,1}$	$Escore_{2,2}$	⋯	$Escore_{2,m}$
⋮	⋮	⋮	⋱	⋮
$F(w_\eta, k_w)$	$Escore_{n,1}$	$Escore_{n,2}$	⋯	$Escore_{n,m}$

FIGURE 4. MII.

- **GenBF**$_{\mathbf{q_w}}(\mathbf{q_w}, \mathbf{K}) \rightarrow \mathbf{T_2}$. For the wildcard keyword q_w in Q, DU generates the corresponding BF trapdoor as follows:

1. For the query wildcard keyword q_w, it can be represented as $q_w = q_w[1], q_w[2], ..., q_w[i], , q_w[i+2], ..., q_w[j], , q_w [j+2], .., q_w[h]$;
2. Initialize Bloom filter BF_{q_w} for q_w;
3. Generate a collection $S_{trapdoor} = \{q_w[1]||1, q_w[2]||2, ..., q_w[i]||i, q_w[j+2]||(-(h-j-1)), ..., q_w[h]||(-1), q_w[1]|0, q_w[2]||0, ..., q_w[i]||0, q_w[i+2]||0, ..., q_w[j]||0, q_w[j+2]||0, ..., q_w[h]||0\}$. In other words, the set S contains the positive order of the characters before the first wildcard, the reverse order of the characters after the last wildcard and the existence of all characters except the wildcard.

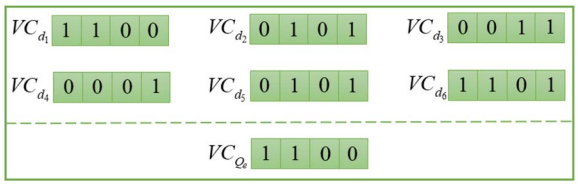


FIGURE 5. DCB-vectors and QCB-vector.

4. For each element s in $S_{trapdoor}$, calculate its corresponding PRF values $\{F(s,k_i)\}_{1\le i\le r}$ and add the r values of each element in $S_{trapdoor}$ to the BF_{q_w}, namely the positions of the r values are set to 1.

We denote the BF_{q_w} as trapdoor T_2.

- **GenPRFValue**$(\mathbf{Q_e}, \mathbf{k_w}) \rightarrow \mathbf{T_3}$. For each $q \in Q_e$, calculate corresponding PRF value $F(q,k_w)$. The trapdoor $T_3 = \{(F(q_1,k_w), F(q_2,k_w), ..., F(q_l,k_w))\}$. Finally, DU sends $T = \{T_1, T_2, T_3\}$ to the CS.

4.4. *Query* $\rightarrow \langle id, Escore\rangle$

Corresponding to the three-part *Indexs* and three-part *Trapdoors*, we can divide the search process into three stages: First, calculate the inner product of each DCB-vector and the QCB-vector according to the index I_1 and trapdoor T_1 to filter out the documents that are entirely impossible to match, i.e. the identifiers set CID of the candidate documents can be obtained. Second, use CBF-Tree and BF_{q_w} to search for the PRF Values of the keywords that match with q_w. Finally, calculate the encrypted relevance score of each document in the CID according to I_3, T_3 and the search result for step 2 and send these scores to the user. DU decrypts these scores to obtain the plaintext relevance scores and the ranking result.

The algorithm includes the following three sub algorithms.

- **Filtering**$(\mathbf{I_1}, \mathbf{T_1}) \rightarrow \mathbf{CID}$. Initialize a set $CID = \emptyset$. With the trapdoor T_1 and the index I_1, CS computes the inner product of each VC_{d_i} and VC_{Q_e} as described in the Equation 5. We can see that calculating the inner product of encrypted vectors is equivalent to calculating the inner product of unencrypted vectors. For each $VC_{d_i} \in VC_D$, if $VC_{d_i} \cdot VC_{Q_e} \neq \{0\}^\tau$, then id_i is added in the identifier set CID. Namely, $CID = \{id_i | VC_{d_i} \cdot VC_{Q_e} \neq \{0\}^\tau \bigwedge id_i \in ID\}$.

$$\begin{aligned} I_1 \cdot T_1 &= (M_1^T VC_{d_i1}) \cdot (M_1^{-1} VC_{Q_e1}) + (M_2^T VC_{d_i2}) \\ &\quad \cdot (M_2^{-1} VC_{Q_e2}) \\ &= (M_1^T VC_{d_i1})^T (M_1^{-1} VC_{Q_e1}) + (M_2^T VC_{d_i2})^T (M_2^{-1} VC_{Q_e2}) \\ &= VC_{d_i1}^T M_1 M_1^{-1} VF_{Q_e1} + VF_{d_i2}^T M_2 M_2^{-1} VF_{Q_e2} \\ &= VC_{d_i1} \cdot VC_{Q_e1} + VF_{d_i2} \cdot VF_{Q_e2} \\ &= VC_{d_i} \cdot VC_{Q_e} \end{aligned} \tag{5}$$

We explain this algorithm by **Example1**. We assume $Q_e = \{w_1, w_8, w_{10}\}$. In Fig. 5, we present the specific DCB-vectors and QCB-vector according to **GenDCBVector** and **GenQCBVector**. Therefore, by calculating the inner product of each VC_{d_i} and VC_{Q_e}, we can get $CID = \{id_1, id_2, id_5, id_6\}$.

- **Search**$(\mathbf{I_2}, \mathbf{T_2}) \rightarrow \mathbf{F}(\mathbf{w}_\eta, \mathbf{k_w})$. We follow the following steps to search for keywords that match with the keyword q_w:

1. Starting from the root node, breadth first traversal is performed on CBF-Tree. On each node i traversed, check whether the BF of the node i in which all bits are set to 1 in BF_{q_w} are 1;
2. If not, ignore all nodes in the subtree whose root node is i in the following search, since there are no keywords matching q_w in this subtree.
3. Otherwise, go on to search from node i until all leaf nodes whose corresponding BF in which all bits are set to 1 in BF_{q_w} are 1 are searched.

Here, we assume that q_w matches only one keyword in the keyword dictionary. For the case of matching more than one keyword, we use the same method to calculate and finally take the maximum score value in all possible cases. We mark the identifier of the matched keyword as w_η. That is, CS will search for the keyword PRF Value $F_w\left(w_\eta, k_w\right.$.

- **EscoreCalculate**$(\mathbf{I_3}, \mathbf{T_3}, \mathbf{F}(\mathbf{w}_\eta, \mathbf{k_w})) \rightarrow \langle \mathbf{id}, \mathbf{Escore}\rangle$. For every $id_i \in CID$, CS calculates the encrypted relevance score of the corresponding document d_i as

$$Escore_i = \sum_{\{k|F(w_k,k_w)\in T_3\}} Escore_{k,i} + Escore_{\eta,i} \tag{6}$$

Therefore, CS can get the ID-Escore pairs list

$$\langle id, Escore\rangle = \{\langle id_i, Escore_i\rangle\}_{id_i \in CID}$$

Finally, CS sends the $\langle id, Escore\rangle$ and $F(w_\eta, k_w)$ to DU.

4.5. *Recovery* $\rightarrow \langle id, Score\rangle$

The details of the algorithm are as follows.

- **Recovery**$(\mathbf{Escore}, \mathbf{Q_e}, \mathbf{F}(\mathbf{w}_\eta, \mathbf{k_w}), \mathbf{k_w}, \mathbf{k_s}) \rightarrow \langle \mathbf{id}, \mathbf{Score}\rangle$. When DU receives $F(w_\eta, k_w)$, he or she firstly uses the key k_w to decrypt $F(w_\eta, k_w)$ to obtain the keyword w_η that matches q_w. Then, use the key k_s to decrypt the encrypted relevance score to obtain the plaintext

relevance score

$$Score_i = Escore_i - \sum_{\{j|q_j \in Q_e\}} F(j||id_i, k_s) - F(\eta||id_i, k_s) \tag{7}$$

Therefore, DU can get the ID-Score pairs list

$$\langle id, Score\rangle = \{\langle id_i, Score_i\rangle\}_{id_i \in CID}$$

Then, DU can rank the documents according to the relevance score and choose parameter k to request the CS to return the top-k encrypted documents. Finally, DU decrypts these documents with sk to obtain the corresponding plain documents.

5. SECURITY ANALYSIS

We first describe the leakage profile of MRSW, and then give the security theorem to show that this is all of the information leaked by the scheme.

· (*Leakage function* $\mathcal{L}_1$) Given the set of documents D,

$$\mathcal{L}_1(D) = \left\{\#D, [id_i]_{i=1}^{\#D}, |c_i|_{i=1}^{\#D}, \#W, |BF|, \#BF[1], \tau, [\#C_j]_{j=1}^{\#CL}\right\}$$

where $\#D$ represents the number of documents, id_i denotes the identifier of document d_i, $|c_i|$ denotes the size of encrypted document c_i, $\#W$ represents the number of keywords contained in keyword dictionary, $|BF|$ denotes the size of BF, $\#BF[1]$ is the number of occurrences of bit 1 in the BF, τ represents the number of keyword clusters and $\#C_j$ represents the number of keywords in each cluster.

· (*Leakage function* $\mathcal{L}_2$) For the document set D and query Q,

$$\mathcal{L}_2(D, Q) = \left\{\tau, \#BF_{q_w}[1], S_p(Q), A_p(D, Q)\right\}$$

where τ denotes the number of keyword clusters, $\#BF_{q_w}[1]$ denotes the number of occurrences of bit 1 in the BF_{q_w}, $S_p(Q)$ and $A_p(D, Q)$ denote the search pattern and access pattern, respectively.

We introduce the definitions of search pattern and access pattern as follows:

DEFINITION 5.1. *(Search Pattern, S_p). Let $Q_i = \{q_i^1, q_i^2, ..., q_i^l, q_i^{l+1}\}$ be the i-th query keyword set, $\left\{(q_1^1, q_1^2, ..., q_1^l, q_1^{l+1}), (q_2^1, q_2^2, ..., q_2^l, q_2^{l+1}), ..., (q_\delta^1, q_\delta^2, ..., q_\delta^l, q_\delta^{l+1})\right\}$ be the keyword set for δ consecutive queries and $i[j]$ represents the j-th keyword of i-th query. Then, $S_p[i[j], p[r]] = 1$ if $q_i^j = q_p^r$ and $S_p[i[j], p[r]] = 0$ otherwise for $1 \le i, p \le \delta$ and $1 \le j, r \le l+1$.*

DEFINITION 5.2. *(Access Pattern, A_p). For query Q, the access pattern is definied as $E(Q) = \left\{\langle d_1, Escore_1\rangle, \langle d_2, Escore_2\rangle, ..., \langle d_s, Escore_s\rangle\right\}$, which is the identifier set of candidate document and its encrypted relevance score pair corresponding to the search query.*

THEOREM 5.1. *If SKE is PCPA-secure and F is a secure PRF, then MRSW satisfies Multi-keyword rank CKA2-security.*

Proof We prove the CKA2-security by showing the existing of a PPT simulator $\mathcal{S}$ such that for each PPT adversary $\mathcal{A}$, the outputs of $\mathbf{Real}_{\mathcal{A}}^{\Pi}(\mathbf{n})$ and $\mathbf{Ideal}_{\mathcal{A},\mathcal{S}}^{\Pi}(\mathbf{n})$ are indistinguishable.

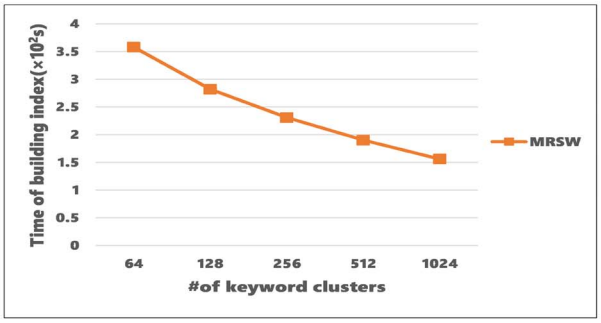
[**Setup**] Given $\mathcal{L}_1(D) = \left\{\#D, [id_i]_{i=1}^{\#D}, |c_i|_{i=1}^{\#D}, \#W, |BF|, \#BF[1], \tau, [\#C_j]_{j=1}^{\#CL}\right\}$, $\mathcal{S}$ simulates the encrypted document set and index as below. For encrypted documents, since SKE is PCPA-secure, $\mathcal{S}$ only needs to choose m random values $\left\{c_1^*, c_2^*, ..., c_m^*\right\}$ such that $|c_1^*| = |c_1|, |c_2^*| = |c_2|, ..., |c_m^*| = |c_m|$. For index I_1, since secure KNN is known plaintext attack (KPA) secure, the index I_1 is secure under the known ciphertext model. For each document d_i, $\mathcal{S}$ generates two random τ-dimensional vectors $VC_{d_i,1}$, $VC_{d_i,2}$ and let $I_1 = \left\{VC_{d_i,1}, VC_{d_i,2}\right\}_{i=1}^{m}$. For index I_2, since the function F that generates the keyword identifier is pseudo-random, $\mathcal{S}$ chooses n random numbers $\left\{R_1, R_2, .., R_n\right\}$ of n-bits as the identifier of the keywords. In addition, $\mathcal{S}$ initializes a BF for each keyword. Since F is a PRF, for each BF, $\mathcal{S}$ randomly chooses $\#BF[1]$ positions and sets the bit of these positions to 1. Then cluster the keywords and generate the corresponding *CBF-Tree* as index I_2. For index I_3, since the identifiers of the documents can be obtained from the $\mathcal{L}_1(D)$, $\mathcal{S}$ can simply copy the list of identifier. The identifiers of keywords have been obtained during the generation of I_2. Since the function F that generates the encrypted relevance score is a PRF, $\mathcal{S}$ generates a n-bit random number $R_{i,j} (1 \le i \le n, 1 \le j \le m)$ between each keyword and each document. Therefore, the index I_3 has been constructed.

[**Simulating Search**] Given $\mathcal{L}_2(D, Q) = \left\{\tau, \#BF_{q_w}[1], S_p(Q), A_p(D, Q)\right\}$, $\mathcal{S}$ simulates search process as follows. According to the S_p, $\mathcal{S}$ first checks whether the encrypted query clustering bit vector has appeared in the previous query, and if so, generates the same encrypted query clustering vector; otherwise randomly generates two τ-dimensional vectors as the encrypted query clustering vector. Then, $\mathcal{S}$ checks whether the wildcard keyword q_w has been queried previously, and if so, $\mathcal{S}$ sets the corresponding BF trapdoor to be the same as before; otherwise, $\mathcal{S}$ randomly chooses $\#BF_{q_w}[1]$ positions of BF_{q_w} and sets the bit of these positions to 1. Finally, $\mathcal{S}$ checks whether the exact keyword has been queried; if so, it sets the keyword identifier to match random numbers previously used; otherwise, $\mathcal{S}$ generates a n-bit random number R as the identifier of the keyword.

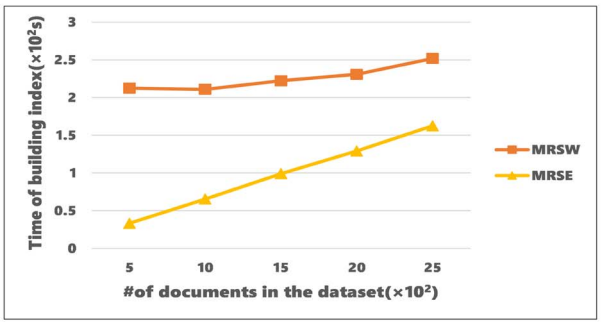
In short, the indistinguishability between the $\mathbf{Real}_{\mathcal{A}}^{\Pi}(\mathbf{n})$ and $\mathbf{Ideal}_{\mathcal{A},\mathcal{S}}^{\Pi}(\mathbf{n})$ is obtained by the PCPA-secure of SKE, KPA-secure of secure KNN and the indistinguishability of PRF from random functions.

TABLE 3. Default values of parameters.

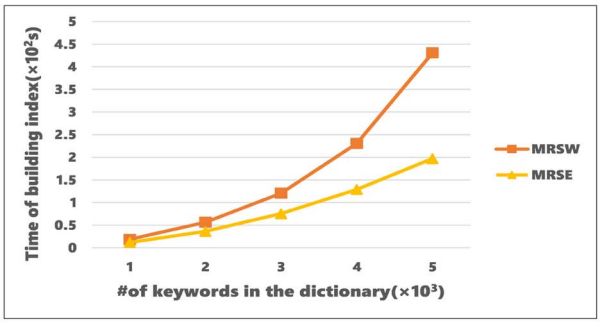
Parameters	τ	m	n	$l+1$
Values	256	2000	4000	16



(a) Index build time vs τ

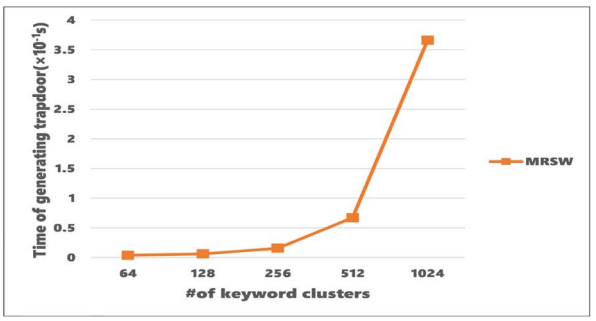


(b) Index build time vs m

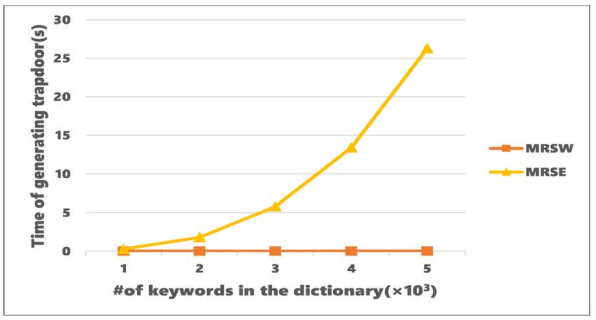


(c) Index build time vs n

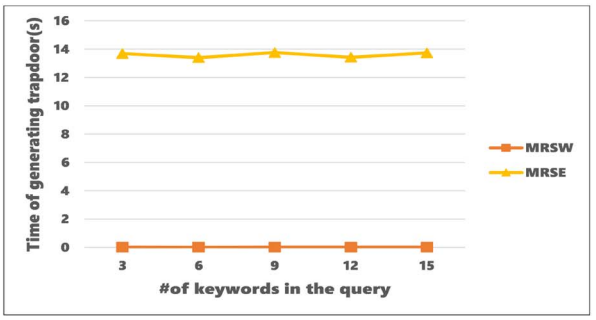
FIGURE 6. Time cost of building index.



(a) Trapdoor generation time vs τ



(b) Trapdoor generation time vs n



(c) Trapdoor generation time vs $l+1$

FIGURE 7. Time cost of generating trapdoor.

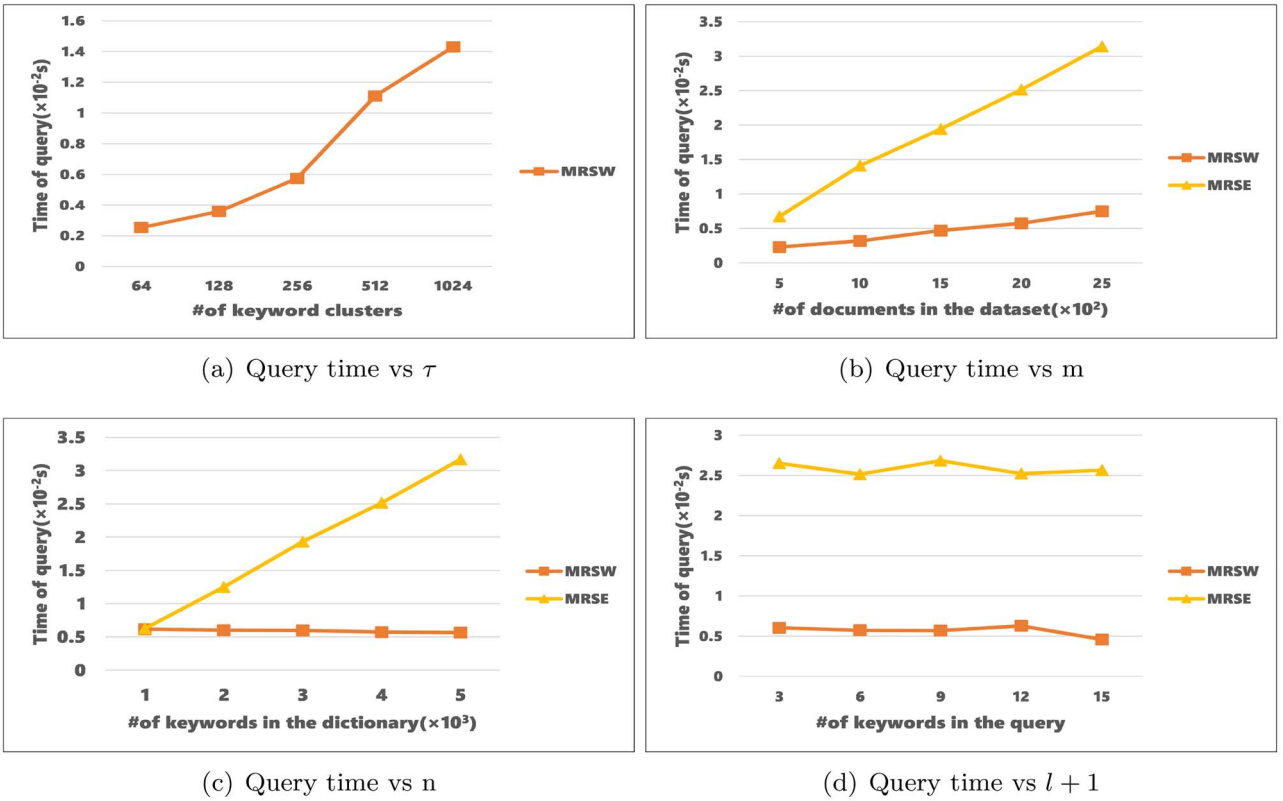
6. PERFORMANCE ANALYSIS

We estimate the performance of MRSW and compare it with the MRSE_I_TF presented in [6]. The whole experiment is run on centos-release-7-8.2003.0.el7.centos.x86_64 operating system and is implemented by 8.3.1 (Red Hat 8.3.1-3) (GCC) compile on a Linux Server with Intel(R) Xeon(R) Gold 6132 CPU@ 2.60 GHZ and 20.0G memory. The data set is built from the Web of Science, including about 2500 documents and 5000 keywords extracted from these documents.

In Table 3, we list the required default parameters, where $\tau, m, n, l+1$, respectively, represent the number of clusters, documents, keywords and query keywords. Next, we use the control variable method to evaluate the influence of the above parameters on the time cost of index building, generating trapdoor and query (We note that there is no clustering of keywords in MRSE, so we only evaluate the influence of τ on MRSW).

6.1. Index Construction

Fig. 6 is used to describe the efficiency of building index with different conditions, including keyword clusters (τ), the number of documents (m) and the number of keywords in the



(a) Query time vs τ (b) Query time vs m

(c) Query time vs n (d) Query time vs $l+1$

FIGURE 8. Time cost of query.

dictionary (n). We did not test the effect of the number of query keywords ($l+1$) on the building index time, because the index building is not influenced by $l+1$.

Fig. 6a displays as τ increases, the time of building index of MRSW decreases. Since we adopt a 'bottom-up' aggregation strategy hierarchical clustering algorithm, it is evident that the more clusters, the less time it takes.

Fig. 6b indicates that when m is fixed, the time of building index of MRSW is longer than that of MRSE. And the time of MRSE is almost linear with m. However, the growth rate of MRSW is much lower than that of MRSE. This is because in MRSE, the time of building index of each document is fixed, but the time of building index of MRSW mainly depends on τ. Therefore, when the document increases to a certain number, the time of building index of MRSE will exceed MRSW.

Fig. 6c shows as n increases, the time of building index of MRSE and MRSW both increases. This is because the dimension of index vector of MRSE and the clustering time of MRSW both increase.

6.2. Trapdoor Generation

Fig. 7 describes the influence of different parameters, including keyword clusters (τ), the number of keywords in the dictionary (n) and the number of query keywords ($l+1$), on the efficiency of generating trapdoor. Since in MRSE and MRSW, trapdoors are generated only in accordance with the keyword dictionary and query keywords, it has nothing to do with m, so we did not test the trapdoor generation time under the different m.

From Fig. 7a, we can see with the growth of τ, the trapdoor generation time of MRSW increases. This is because the number of keyword clusters is the dimension of trapdoor vector and as the dimension increases, the trapdoor generation time naturally increases.

From Fig. 7b, we can see that the trapdoor generation time of MRSE is greatly affected by n. However, the trapdoor generation time of MRSW remains stable and is close to 0 as the n increases. This is exactly consistent with the theoretical analysis. When n increases, the query vector's dimension of MRSE also extends, but the query vector's size of MRSW is equal to τ, which is unchanged.

From Fig. 7c, we can observe the change of n almost no effect on the time cost of trapdoor generation of both MRSE and MRSW. This is because the difference in the cost of generating trapdoors in the two schemes is primarily affected by the difference in the dimensionality of query vector. However, the cost of MRSE to generate trapdoors is about 800 times that of MRSW, which is significant of MRSW.

6.3. Query

Fig. 8 describes the change of the query time when the keyword clusters (τ), the number of documents (m), the number of keywords in the dictionary (n) and the number of query keywords ($l+1$) change, respectively.

Fig. 8a shows the query time of MRSW increases as τ increases. The reason is that the dimension of the index vector and the number of layers of CBF-Tree both increase.

Fig. 8b presents as m goes up, the query time cost of both MRSE and MRSW increase. However, the query time cost and the cost growth rate of MRSW are significantly smaller than those of MRSE. Because in MRSE, the relevance scores of all documents need to be calculated, while in MRSW only documents in the candidate document set are required.

Fig. 8c displays parameter n has little impact on the time cost of query in MRSW. But n has a significant impact on MRSE. The reason is the same as Fig. 7b.

Fig. 8d shows similar trapdoor generation as in Fig. 7c, $l+1$ also almost has no impact on the time of query of these two schemes and the query efficiency of MRSW has a great advantage over MRSE. The reason is also the same as in Fig. 7c.

7. CONCLUSION

We proposed an MRSE system MRSW. The query keyword set may contain a wildcard keyword, which is more in line with the user's actual search needs. To reduce the possibilities of statistical attack of the server, an MII table is generated to enable the ranking to be performed on the user-side. Through rigorous theoretical analysis, the security of MRSW under adaptive chosen-keyword attacks model is proved. Experiments on a real data set show the high efficiency of our MRSW.

DATA AVAILABILITY

The data underlying this article will be shared on reasonable request to the corresponding author.

CONFLICT OF INTEREST

Authors have no conflict of interest to declare.

FUNDING

This work is supported by the National Natural Science Foundation of China (No.62072276, No.61772311).

REFERENCES

[1] Song, D.X., Wagner, D. and Perrig, A. (2000) Practical techniques for searches on encrypted data. In *Proceeding 2000 IEEE Symposium on Security and Privacy. S&P 2000*, pp. 44–55. IEEE, Berkeley, CA, USA 14-17 May.

[2] Goh, E.-J.et al. (2003) Secure indexes. *IACR Cryptol. ePrint Arch.*, 2003, 216.

[3] Chang, Y.-C. and Mitzenmacher, M. (2005) Privacy preserving keyword searches on remote encrypted data. In *International conference on applied cryptography and network security*, pp. 442–455. Springer Berlin Heidelberg, New York, USA June.

[4] Curtmola, R., Garay, J., Kamara, S. and Ostrovsky, R. (2011) Searchable symmetric encryption: improved definitions and efficient constructions. *Journal of Computer Security*, 19, 895–934.

[5] Ning, C., Cong, W., Ming, L., Ren, K. and Lou, W. (2011) Privacy-preserving multi-keyword ranked search over encrypted cloud data. In *INFOCOM 2011. 30th IEEE International Conference on Computer Communications, Joint Conference of the IEEE Computer and Communications Societies*, pp. 829–837. IEEE, Shanghai, China 10-15 April.

[6] Cao, N., Wang, C., Li, M., Ren, K. and Lou, W. (2013) Privacy-preserving multi-keyword ranked search over encrypted cloud data. *IEEE Transactions on parallel and distributed systems*, 25, 222–233.

[7] Yu, J., Lu, P., Zhu, Y., Xue, G. and Li, M. (2013) Toward secure multikeyword top-k retrieval over encrypted cloud data. *IEEE transactions on dependable and secure computing*, 10, 239–250.

[8] Chen, C., Zhu, X., Shen, P., Hu, J., Guo, S., Tari, Z. and Zomaya, A.Y. (2015) An efficient privacy-preserving ranked keyword search method. *IEEE Transactions on Parallel and Distributed Systems*, 27, 951–963.

[9] Xia, Z., Wang, X., Sun, X. and Wang, Q. (2015) A secure and dynamic multi-keyword ranked search scheme over encrypted cloud data. *IEEE transactions on parallel and distributed systems*, 27, 340–352.

[10] Guo, C., Zhuang, R., Chang, C.-C. and Yuan, Q. (2019) Dynamic multi-keyword ranked search based on bloom filter over encrypted cloud data. *IEEE Access*, 7, 35826–35837.

[11] Wang, B., Yu, S., Lou, W. and Hou, Y.T. (2014) Privacy-preserving multi-keyword fuzzy search over encrypted data in the cloud. In *IEEE INFOCOM 2014-IEEE Conference on Computer Communications*, pp. 2112–2120. IEEE, Toronto, ON, Canada 27 April-2 May.

[12] Fu, Z., Wu, X., Guan, C., Sun, X. and Ren, K. (2016) Toward efficient multi-keyword fuzzy search over encrypted outsourced data with accuracy improvement. *IEEE Transactions on Information Forensics and Security*, 11, 2706–2716.

[13] Cao, J., Zhu, J., Lin, L., Xue, Z., Ma, R. and Guan, H. (2019) A novel fuzzy search approach over encrypted data with improved accuracy and efficiency. *arXiv preprint*, **arXiv:1904.12111**.

[14] Hu, C. and Han, L. (2016) Efficient wildcard search over encrypted data. *International Journal of Information Security*, 15, 539–547.

[15] Yang, Y., Liu, X., Deng, R.H. and Weng, J. (2017) Flexible wildcard searchable encryption system. *IEEE Transactions on Services Computing*, 13, 464–477.

[16] Sedghi, S., Van Liesdonk, P., Nikova, S., Hartel, P. and Jonker, W. (2010) Searching keywords with wildcards on encrypted data. In *International Conference on Security and Cryptography for Networks* [S.l.], September, pp. 138–153. Springer Berlin, Heidelberg.

[17] Bösch, C., Brinkman, R., Hartel, P. and Jonker, W. (2011) Conjunctive wildcard search over encrypted data. In *Workshop on Secure Data Management* [S.l.], September, pp. 114–127. Springer Berlin, Heidelberg.

[18] Suga, T., Nishide, T. and Sakurai, K. (2012) Secure keyword search using bloom filter with specified character positions. In *International Conference on Provable Security* [S.l.], September, pp. 235–252. Springer Berlin, Heidelberg.

[19] Zhao, F. and Nishide, T. (2016) Searchable symmetric encryption supporting queries with multiple-character wildcards. In *International Conference on Network and System Security* Cham, 28 September, pp. 266–282. Springer International Publishing.

[20] Bloom, B.H. (1970) Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13, 422–426.

[21] Chase, M. and Kamara, S. (2010) Structured encryption and controlled disclosure. In *International conference on the theory and application of cryptology and information security* Singapore, 5-9 December, pp. 577–594. Springer, Berlin, Heideberg.