Privacy-Preserving Multi-keyword Top-k Similarity Search Over Encrypted Data

Xiaofeng Ding, *Member, IEEE,* Peng Liu and Hai Jin, *Senior Member, IEEE*

Abstract—Cloud computing provides individuals and enterprises massive computing power and scalable storage capacities to support a variety of big data applications in domains like health care and scientific research, therefore more and more data owners are involved to outsource their data on cloud servers for great convenience in data management and mining. However, data sets like health records in electronic documents usually contain sensitive information, which brings about privacy concerns if the documents are released or shared to partially untrusted third-parties in cloud. A practical and widely used technique for data privacy preservation is to encrypt data before outsourcing to the cloud servers, which however reduces the data utility and makes many traditional data analytic operators like keyword-based top-k document retrieval obsolete. In this paper, we investigate the multi-keyword top-k search problem for big data encryption against privacy breaches, and attempt to identify an efficient and secure solution to this problem. Specifically, for the privacy concern of query data, we construct a special tree-based index structure and design a random traversal algorithm, which makes even the same query to produce different visiting paths on the index, and can also maintain the accuracy of queries unchanged under stronger privacy. For improving the query efficiency, we propose a group multi-keyword top-k search scheme based on the idea of partition, where a group of tree-based indexes are constructed for all documents. Finally, we combine these methods together into an efficient and secure approach to address our proposed top-k similarity search. Extensive experimental results on real-life data sets demonstrate that our proposed approach can significantly improve the capability of defending the privacy breaches, the scalability and the time efficiency of query processing over the state-of-the-art methods.

Index Terms—Cloud computing, privacy preserving, data encryption, multi-keyword top-k search, random traversal

1 Introduction

Cloud computing has emerged as a disruptive trend in both IT industries and research communities recently, its salient characteristics like high scalability and pay-as-you-go fashion have enabled cloud consumers to purchase the powerful computing resources as services according to their actual requirements, such that cloud users have no longer need to worry about the wasting on computing resources and the complexity on hardware platform management [1], [2]. Nowadays, more and more companies and individuals from a large number of big data applications have outsource their data and deploy their services into cloud servers for easy data management, efficient data mining and query processing tasks.

But when the companies and individuals enjoy these advantages in cloud computing, they also need to take the privacy concern of the outsourced data into account. Because data sets in many applications often contain sensitive information like e-mails, electronic health records and financial transaction records, when the data owner outsourcing such sensitive data to the cloud servers which are considered to be partially trusted, the data can be easily accessed and analyzed by cloud service providers illegally. Since the analysis of these data sets may provide profound insights into a number of key areas in society (such as e-research, healthcare, medical and government services), thus data owners need effective, scalable and privacy-preserving services before releasing their data to the cloud.

Xiaofeng Ding, Peng Liu and Hai Jin are with the Services Computing Technology and System Lab, Cluster and Grid Computing Lab, the School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, China.
E-mail: {xfding, liup, hjin}@hust.edu.cn.

Data encryption has been widely used for data privacy preservation in data sharing scenarios, it refers to mathematical calculation and algorithmic scheme that transform plaintext into cyphertext, which is a non-readable form to unauthorized parties. A variety of data encryption models have been proposed [3], [4], [5] and they are used to encrypt the data before outsourcing to the cloud servers. However, applying these approaches for data encryption usually cause tremendous cost in terms of data utility, which makes traditional data processing methods that are designed for plaintext data no longer work well over encrypted data.

The keyword-based search is such one widely used data operator in many database and information retrieval applications, and its traditional processing methods cannot be directly applied to encrypted data. Therefore, how to process such queries over encrypted data and at the same time guarantee data privacy becomes a hot research topic. Fortunately, many methodologies based on searchable encryption have been studied. For example, [6], [7], [8] deal with the single keyword search, and works [9], [10], [11], [12], [13] support the multi-keyword boolean search. However, the single keyword search is not smart enough to support advanced queries and the boolean search is unrealistic since it causes high communication cost. Therefore, more recent works like [14], [15], [16] focus on the multi-keyword ranked search, which is more practical in pay-as-you-go cloud paradigm. But most of these methods cannot meet the high search efficiency and the strong data security simultaneously, especially when applying them to big data encryption poses great scalability and efficiency challenges.

Motivated by this, in this paper, we focus on a special type of multi-keyword ranked search, namely the multi-keyword top-k search, which has been a very popular database operator in many important applications, and only needs to return the k documents with the highest relevance scores. For supporting multi-keyword search, we introduce the vector space model which represents documents and queries as vectors. In order to support top-k search, the relevance scores between documents and queries should be calculated, therefore, the TF$\times$IDF (term frequency $\times$ inverse document frequency) model is introduced as a weighting rule to compute the relevance scores for ranking purposes.

In addition, to improve the query efficiency for better user experiences, we propose a group multi-keyword top-k search scheme (GMTS), which is based on partition and supports top-k similarity search over encrypted data. In this scheme, the data owner divides the keywords in the dictionary (suppose that the dictionary contains all the keywords that could be extracted from all documents) into multiple groups and establishes a searchable index for each group. On the other side, to better control the size of indexes, we adopt *champion lists* [17], [18] into our scheme, where the index of a keyword group only stores the top-ck documents of the corresponding keyword (the top-ck documents of a keyword represent the $c * k$ documents that have the highest relevance scores to this keyword, where c is a positive integer). Furthermore, we propose a random traversal algorithm (RTRA) to strengthen the data security, where the data owner builds a binary tree as searchable index and assigns a random switch to each node, so the data user can assign a random key to each query. Therefore, the data user can change the results and visiting paths of queries by using different keys, which maintains high accuracy of queries. Finally, we combine the GMTS and the RTRA together into an efficient and secure solution to our proposed problem.

Our contributions can be summarized as follows:

- We first propose the random traversal algorithm which makes the cloud server randomly traverse on index and returns different results for the same query, and in the meantime, it maintains the accuracy of queries unchanged for higher security.
- Based on the random traversal algorithm, we present one both efficient and secure searchable encryption scheme, which can support top-k similarity search over encrypted data. In this scheme, the data owner can control the level of query unlinkability without sacrificing accuracy.
- Our experimental results show that our methods are more efficient than the state-of-the-art methods and can better protect data privacy. Especially, our proposed method has good scalability performance when dealing with large data sets.

The rest of this paper is organized as follows: In Section 2, we review and analyze the related works. Section 3 introduces the system model and threat model, the preliminaries and our design goals. The random traversal algorithm is introduced in Section 4. Section 5 describes the framework of the GMTS and describes the RGMTS framework in Section 6. Section 7 shows and analysis our experimental results. Section 8 concludes this paper.

2 Related Work

Searchable encryption (SE) is a hot research field, especially with the emergence of cloud computing. In this section, we review and analyze the existing searchable encryption schemes. SE can be divided into public key searchable encryption [4], [9], [19], [20] and symmetric searchable encryption (SSE) [3] [7], [8] according to different cryptography primitives. In this paper, we focuses on the symmetric searchable encryption because public key searchable encryption usually are computationally expensive [15], [21].

Abundant works [3], [7], [8], [22], [23] are proposed to deal with symmetric searchable encryption. Song et al. [6] first defined the problem of searching on encrypted data and proposed a symmetric searchable encryption scheme with linear complexity. After that, Goh et al. [7] formulated a security definition for SSE and proposed a secure index which is based on pseudo-random functions and Bloom filters, but the time cost of Goh's scheme is $O(n)$. Curtmola et al. [3] introduced two formal definitions of SSE and proposed a method which is based on inverted list to improve the query performance, their method is proved to be more efficient than other works. However, most of these works can only support single keyword boolean search, which is not advanced enough to support complex functionalities. In recent years, many works have been proposed to achieve different kinds of complex queries like similarity search, multi-keyword search, etc. In general, the literatures [22] and [24] used wildcard-based techniques, [25] based on B^{ed}-tree and [26], [27], [28], [29] applied the locality sensitive hashing (LSH) to deal with similarity search. Works [10], [11], [12], [13] support multi-keyword boolean search, but boolean search is inefficient because it returns all the documents that satisfy the query criteria. Hence, some recent works are proposed to deal with the bandwidth-saving multi-keyword ranked search [14], [15], [16], [23], [30].

Cao et al. [15] proposed the multi-keyword ranked search over encrypted data for the first time and built a searchable index based on the vector space model, and chosen "coordinate matching" to measure the similarity between queries and documents. However, in their schemes, the time complexity of search is $O(nm)$ (n is the number of keywords in dictionary, m is the size of the documents that stored in the cloud server), and the time complexity of trapdoor construction is also very high. Sun et al. [14] proposed a tree-based index structure which is based on the vector space model and the TF$\times$IDF model. This structure achieves sub-linear time complexity, but it is vulnerable in protecting data privacy. One step further, Xia et al. [16] proposed a Greedy Depth-first Search tree-based searchable encryption scheme EDMRS, which achieved more efficiency than early works, but the cost of search remains high and the time complexity of creating trapdoor is high $O(n^2)$.

The works [14], [15], [16] add random numbers ξ_j in indexes or queries to disturb the relevance scores between queries and documents, and they claimed that the value of $\sum \xi_j$ can be adjusted to control the level of query unlinkability. But they can not protect the query unlinkability thoroughly, because in order to guarantee the accuracy of queries, the level of query unlinkability is usually get limited. Actually, the cloud server can easily link two identical

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2017.2693969, IEEE Transactions on Dependable and Secure Computing

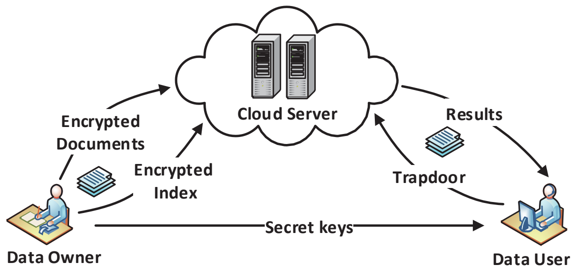


Fig. 1. The system model of searching over outsourced encrypted data.

queries by analyzing and comparing the results and visited paths. For instance, if the data user submits two identical queries to the cloud server, and sets the correct ratio to 80%. Even though the relevance scores are disturbed by adding random numbers, it is expected that at least 60% of the results and visited paths are the same.

3 Problem Formulation

3.1 System Model

As shown in Fig. 1, the system model we considered in this paper contains three parts: the data owner, the data user and the cloud server. The data owner uploads document collection D to the cloud server, but this collection may contain sensitive information. To protect data privacy, the data owner has to encrypt D before outsourcing it to the cloud server. Furthermore, in order to enable the cloud server to process query efficiently over the encrypted document collection C, the data owner constructs an encrypted searchable index I_e locally. Finally, the data owner outsources both the encrypted document collection C and the encrypted searchable index I_e to cloud, and shares the secret key of trapdoor generation and document decryption to authorized data users with secure channels.

When the data user wants to search with a query, s/he generates the trapdoor T for this query firstly by query encryption, and then submits the trapdoor to cloud server for query processing. After receiving T, the cloud server calculates the relevance scores between trapdoor T and the documents in index I_e, and returns k documents with the highest scores to the data user.

Note that, the search control is outside the scope of our paper. Therefore, similar to works [16], [22], [27], [31], [32], we assume data users are trusted entities and the trapdoors are generated by data users themselves.

3.2 Threat Model

In this paper, we treat data owner and data user as trusted entities, but cloud server is considered to be "honest-but-curious" as adopted in most works on secure cloud data search. The server is honest as it runs the programs and algorithms correctly, it is curious since the cloud service providers can easily access and analyze the encrypted data, and even record queries to learn additional information. Based on the information which can be learned by cloud sever, we consider two threat models as [15], [31].

Known Ciphertext Model. This threat model corresponds to the ciphertext-only attack, as the cloud server only knows the encrypted document collection C, encrypted searchable index I_e and trapdoor T.

Known Background Model. Compared to the known ciphertext model, this model is more stronger, as the cloud server here not only knows the ciphertext of document collection, searchable index and query, it is supposed to have other background knowledge like statistic information about the document collection, which will expose more knowledge to cloud. For instance, when the cloud server know the normalized TF distributions of certain keywords, it can identify these keywords by comparing the normalized TF distributions [14], [15], [16], [30], [33].

TABLE 1
Notations

Notation	Description
D	the plaintext document collection, denoted as $D = \{D_1, D_2, ..., D_m\}$, D_i is a document of D
C	the encrypted document collection, denoted as $C = \{C_1, C_2, ..., C_m\}$
W	the dictionary which contains n keywords which appeared in the document collection D, denoted as $W = \{w_1, w_2, ..., w_n\}$
W_q	the keyword set which is a subset of the dictionary W and contains t keywords that data users want to search
WG	the keyword group, denoted as $WG = \{WG_1, WG_2, ..., WG_b\}$, where each group WG_i contains d keywords
b	the number of groups in the keyword group WG, it means $b = ceiling(n/d)$
I	the unencrypted form of searchable index
I_e	the encrypted form of the searchable index I
Q	the query which is constructed based on the keyword set W_q
T	the trapdoor, which is the encrypted form of query Q
$\widehat{Re}$	the search results that the cloud server returns to data users, denoted as $\widehat{Re} = \{R_1, R_2, ..., R_k\}$
$Score(Q, D_i)$	the relevance score between query Q and document D_i

3.3 Preliminaries

3.3.1 Multi-keyword top-k Search

Let D be the plaintext document collection that the data owner will outsource to cloud servers, and D_i represents a document in D. W is a dictionary and $Score(Q, D_i)$ is the relevance score between query Q and document D_i (the mainly used notations in this paper are summarized in Table 1). The multi-keyword top-k search [17] is used to find the k documents with the highest relevance scores to query Q, the formal definition is given as follows:

Definition 1. *(Multi-keyword Top-k Search) Given a query Q and a document collection D, find k documents $\widehat{Re} = \{R_1, R_2, ..., R_k\}$ in D with the highest relevance scores, this is to say, $\forall D_i \in D/\widehat{Re}, Score(Q, D_i) \leq Score(Q, R_j)(1 \leq j \leq k)$.*

e.g., document collection D has four documents, as shown in Fig. 2, where each document is represented as a vector and these vectors store the TF values of their corresponding keywords. For a query $(0, 0.5, 0, 0, 0.1, 0.6)$, the relevance scores of the documents D_1, D_2, D_3 and D_4 are 0.33, 0.24, 0.12 and 0.63, respectively. It is obvious that

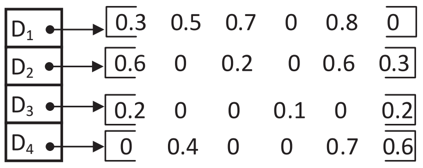


Fig. 2. A document collection, where each document is represented as a vector.

documents D_1 and D_4 are the top 2 documents, as their scores are higher than others.

3.3.2 *Vector Space Model and TF×IDF*

Vector space model [34] and TF×IDF [17] are widely used in information retrieval. In vector space model, each document and query is represented as a vector (but in this paper, each document and query is represented as a group of vectors), which supports multi-keyword search quite well. TF×IDF is used as a weighting function to calculate the similarity between documents and queries. Assume w_i is a keyword in the dictionary W, the term frequency (TF) of w_i in a document D_j is denoted as $TF_{i,j}$, which measures the weighting of keyword w_i in document D_j. Inverse document frequency (IDF) is used to measure the overall importance of a keyword in the entire document collection, we use IDF_i to represent the IDF of keyword w_i. The relevance score between query Q and document D_j is:

$$Score(Q, D_j) = \frac{1}{|D_j|} \sum_{w_i \in Q} TF_{i,j} \cdot IDF_i \tag{1}$$

Here $TF_{i,j} = 1 + \ln f_{j,i}$, where $f_{j,i}$ is the number of times that the keyword w_i occurs in document D_j. And $IDF_i = \ln(1 + \frac{m}{m_i})$, where m_i is the number of documents which contain the keyword w_i. $|D_j|$ denotes the Euclidean length of document D_j, and it can be calculated as:

$$|D_j| = \sqrt{\sum_{w_i \in D_j} (1 + \ln f_{j,i})^2}$$

3.4 Design Goals

Our goals contain three aspects: 1) Supporting multi-keyword top-k similarity search over encrypted data; 2) Search with high efficiency; 3) Privacy-preserving. The details are listed as below:

Multi-keyword top-k Search: To design a searchable encryption scheme that enables the cloud server to support multi-keyword top-k similarity search over encrypted data;

Search efficiency: Our scheme should be efficient in index construction, trapdoor generation and search processing, and it should be more efficient and effective than the state-of-the-art methods;

Privacy-preserving: Our scheme should protect the privacy of indexes and queries at the same time. They are

- *Index security and Query security*: The plaintext information of encrypted searchable index and trapdoor should be protected.
- *Keywords Privacy*: The cloud server cannot identify whether a certain keyword is contained in a query by analyzing indexes or search results.

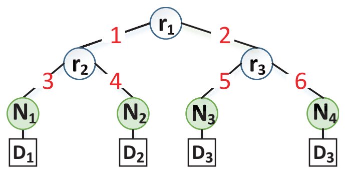


Fig. 3. An example of tree-based index for document collection $D = \{D_1, D_2, D_3\}$.

- *Query Unlinkability and Access Pattern*: The cloud server cannot distinguish whether two identical trapdoors are from the same query, which needs us to hide the visiting paths on index and access pattern of query, where access pattern represents the available information in search results [15].

4 The Random Traversal Algorithm

Fig. 3 is a tree-based index, where N_1, N_2, N_3, N_4 are the leaf nodes. Both the stored value of nodes N_3 and N_4 are document D_3. If the result of a query contains documents D_1 and D_3, then we have to path through edge 1 and 3 to visit D_1. On the contrary, visiting D_3 has two options: from edge 2 to 5 or from edge 2 to 6. Inspired by the above example, we propose a random traversal algorithm (RTRA). In RTRA, giving two identical queries, their visiting paths in index and search results can be different, but maintain the accuracy of queries unchanged. The main idea is as follows: 1) enlarging the whole document collection E times, hence each document in result has E options; 2) assigning a switch to each document; 3) building a tree-based index for the whole document collection, where document identifiers are stored in leaf nodes. 4) assigning a random key to each query. Therefore, data users can control the visiting paths and search results by assigning different keys. Next, we further discuss the details of RTRA.

4.1 RTRA Framework

4.1.1 *Enlarging Document Collection*

Firstly, the documents in collection D are randomly divided into L groups with the same size, and the divided document collection is represented as $DG = \{DG_1, DG_2, ..., DG_L\}$. Then, each document group is copied E times and each document is assigned with a unique document identifier. We use DG^x to represent the enlarged document collection, where $DG^x = \{DG_1^1, .., DG_1^E, ..., DG_L^1, ..., DG_L^E\}$ and DG_i^j represents j-th copy of document group DG_i. After D is enlarged, each document has E copies and were distributed in different groups.

For example, assume $L = 2$, $E = 2$ and document collection D has four documents $D = \{D_1, D_2, D_3, D_4\}$. We divide D into two groups $DG_1 = \{D_1, D_2\}$ and $DG_2 = \{D_3, D_4\}$. After D is enlarged, we get

$$\begin{aligned} DG^x &= \{DG_1^1, DG_1^2, DG_2^1, DG_2^2\} \\ &= \{\{D_1^1, D_2^1\}, \{D_2^2, D_1^2\}, \{D_3^1, D_4^1\}\{D_4^2, D_3^2\}\} \\ &= \{D_1^1, D_2^1, D_2^2, D_1^2, D_3^1, D_4^1, D_4^2, D_3^2\} \end{aligned}$$

Where D_i^1 and D_i^2 are two file copies of document D_i ($1 \leq i \leq 4$).

Notice 1). The order of documents in each group is random (e.g., the order of copy DG_1^1 is D_1^1, D_2^1, while in

DG_1^2 is D_2^2, D_1^2 instead of D_1^2, D_2^2). The reason is that if the order of two group copies are the same, visiting path in the low levels of index may be the same. 2). For disturbing the visiting path of query on index, the order of each group in DG^x is still selected randomly, such as the order of DG^x could be $\{DG_1^1, DG_2^2, DG_2^1, DG_1^2\}$ instead of $\{DG_1^1, DG_1^2, DG_2^1, DG_2^2\}$.

4.1.2 Assigning Switch

Each document of the enlarged document collection DG^x is assigned a switch which is a vector with length r (where $r = L*E$). For switch formulation and describe conveniently, we give the definition of *The Same Switch Form*:

Definition 2. *(The Same Switch Form) Given two nodes N_1 and N_2, if not all bits of their switches are zero, and both $N_1.switch[i]$ and $N_2.switch[i]$ are equal to zero or bigger than zero at the same time (where $i = 1, 2, .., r$), we call the two switches have the same form and the two nodes have the same switch form.*

If two documents belong to the same group and they will be assigned with the switches that have the same form, otherwise they are different. $switch_i^j$ represents the switch of these documents which belong to document group DG_i^j and it is calculated as follows:

$$switch_i^j[\kappa] = \begin{cases} 0 & \text{if } \kappa! = i*E+j \\ \theta + |Rand()|\%\tau & \text{if } \kappa = i*E+j \end{cases} \quad (2)$$

Where $\kappa = 1, 2, ..., r$. θ and τ are two positive integers, and they are set to 5 and 10 in this paper. $Rand()$ is a random number generator and $|Rand()|$ is its absolute value.

4.1.3 Building Index

We build a binary tree I for document collection DG^x as index, where document identifiers are stored in leaf nodes. Let N represents a node in I, and we denote its form as $\langle fid, l_c, r_c, switch\rangle$. If N is a leaf node, fid is the document identifier, l_c and r_c are null. Otherwise, fid is null, l_c and r_c point to its left and right child, respectively. If the children of node N have the same switch form, we add node N to the document group where its children belong to, furthermore, we calculate the switch of this node by Equation 2. Otherwise, all the bits of the switch are set to zero.

4.1.4 Assigning Keys

For getting different visiting paths and search results when processing a query at two different periods, the query is assigned with a random key, where the key is a vector with the same length as switches and represented as key. When generating a key, data user selects one dimension from each E dimensions of key, and the selected dimensions are set to zero, while the others are set to different random negative integers. The reason is that each document has E copies, but when processing search, the cloud server only needs to traverse one copy. The traversed copy is controlled by the values of key, if $key[i*E+j]$ equals to zero then the documents in DG_i^j will be traversed (where i in $[1, L]$ and j in $[0, E-1]$). The value of key is defined as below:

$$key[\kappa] = \begin{cases} -\theta - |Rand()|\%\tau & \text{if } \kappa! = j*E+\omega_j \\ 0 & \text{if } \kappa = j*E+\omega_j \end{cases} \quad (3)$$

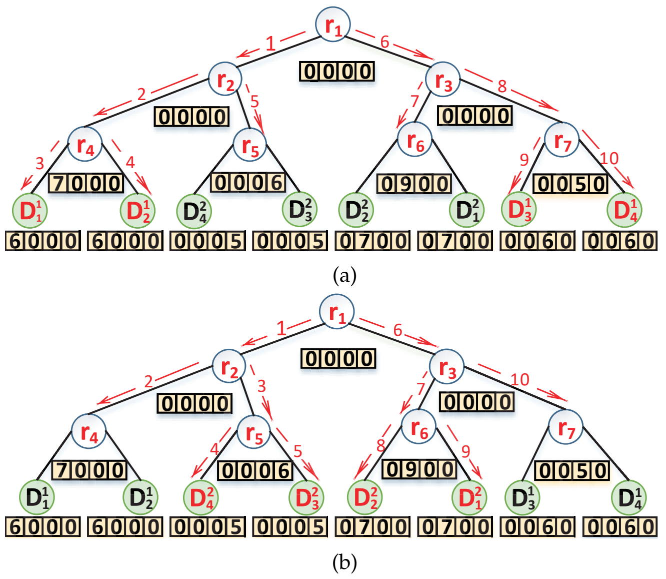


Fig. 4. An example of the random traversal algorithm with document collection $D = \{D_1, D_2, D_3, D_4\}$, $E = 2$ and $L = 2$. The search process starts at the root node r_1 and uses depth-first traversal method to visit all nodes. (a) shows the visiting path and query results with key $[0, -6, 0, -7]$, the search starts from r_1, and first reaches leaf nodes D_1^1 and D_2^1 through r_2 and r_4, because the scores of r_2 and r_4 are equal to 0. Then, the nodes r_5, r_3, r_6, r_7, D_3^1 and D_4^1 are visited. The nodes D_4^2, D_3^2, D_2^2 and D_1^2 are not traversed, because the scores of r_5 and r_6 are less than 0. Finally, we get results $\{D_1^1, D_2^1, D_3^1, D_4^1\}$. (b) shows the visiting path and results of query with key $[-8, 0, -5, 0]$, the results are $\{D_4^2, D_3^2, D_2^2, D_1^2\}$, the visited nodes are $\{r_1, r_2, r_4, r_5, D_4^2, D_3^2, r_3, r_6, D_2^2, D_1^2, r_7\}$.

Where $\kappa = 1, 2, ..., r$, $j = floor(\kappa/E)$. ω_j is a random integer in $[0, E-1]$ which represents the selected dimension by the data user.

For example, if we assume $L = 2$, $E = 2$, and the divided document collection is $DG = \{DG_1, DG_2\}$. Thus, the key is a vector of length 4 and each document has two copies. Where $key[1]$, $key[2]$, $key[3]$ and $key[4]$ are used to control the document groups DG_1^1, DG_1^2, DG_2^1 and DG_2^2, respectively. Such as that when $key[1]$ equals to zero, the document group DG_1^1 will be traversed, and so on. Therefore, there have four possibilities $\{DG_1^1, DG_2^1\}$, $\{DG_1^1, DG_2^2\}$, $\{DG_1^2, DG_2^1\}$, $\{DG_1^2, DG_2^2\}$, and the corresponding keys are $[Z, RN, Z, RN]$, $[Z, RN, RN, Z]$, $[RN, Z, Z, RN]$, $[RN, Z, RN, Z]$, where Z is zero and RN can be any random negative integer.

4.1.5 Query Processing

Search starts from the root to the leaf nodes in the tree. For any node N, only when $key \cdot switch \geq 0$ can the cloud server continues to walk along this node. As shown in Fig. 4, the switch of r_5 is $switch_5 = [0, 0, 0, 6]$ and the switch of node r_7 is $switch_7 = [0, 0, 5, 0]$. If the key of a query is $key_1 = [0, -6, 0, -7]$, then documents D_3^1 and D_4^1 will be traversed, while D_3^2 and D_4^2 will be ignored, because $key_1 \cdot switch_7 = 0$ and $key_1 \cdot switch_5 = -42$. On the contrary, if the key is $key_2 = [-8, 0, -5, 0]$, documents D_3^1 and D_4^1 will not be traversed, but D_3^2 and D_4^2 will be traversed since $key_2 \cdot switch_7 = -25$ and $key_2 \cdot switch_5 = 0$.

Note that, even though documents D_3^1 and D_3^2 have different identifiers, since both of them are copies of original document D_3. Hence, D_3^1 and D_3^2 have the same content. Therefore, for two identical queries with different keys, the cloud server may visit different paths and return different results, but their query accuracies are the same.

4.2 Analysis

We have introduced the unencrypted RTRA, but for the concern of more strict security guarantee, the following analysis is based on the encrypted RTRA (the encryption method will be introduced in next section) such that an adversary cannot distinguish two copies just rely on their ciphertext.

4.2.1 Functionality and Information Leakage

When processing queries in the index, if the switch of one node is zero, then this node will be always traversed by the cloud as its product with any key always equals to zero. On the contrary, if the product is less than zero, this node and its children will be pruned. Therefore, to reduce the number of traversed nodes, we try to keep the neighboring leaf nodes in the index to have the same switch form. Thus when building index, these documents come from one group are assigned with the same switch form. Although the above method may expose the grouping information, the adversaries (cloud servers) cannot identify which copies are from one certain document, since the documents order in each group is randomly assigned. Furthermore, since each switch and key contains a set of different random numbers, thus the product of one key with different switches or the product of one switch with different keys are different. Therefore, adversaries cannot directly identify the copies of one certain document by comparing the products.

In addition, document collection D is divided into L groups and each group is copied E times. Therefore, a data user can generate E^L different keys, thus the expectation of the same document in the query result with different keys is $|\widehat{Re}|/E$, where $|\widehat{Re}|$ is the size of search results. It is obviously that the more number of different documents in two search results, the more difficult for adversaries to judge whether the two queries come from the same request or not. Therefore, the data owner can raise the level of security strength by increasing the value of E.

4.2.2 Space Consumption

The storage space occupied by RTRA is E times of original index, as the value of E increases, the space consumption also becomes larger. Thus, there is a trade-off between space and security, but with the rapid development of computer hardware, the space will not be the main problem. However, an approach to reduce the size of indexes is also introduced in the next section.

5 The Scheme of GMTS

In this section, we first introduce the unencrypted group multi-keyword top-k search scheme (UGMTS), which is designed to support multi-keyword top-k similarity search over encrypted data and to improve query efficiency. Then, we describe the EGMTS which is an encrypted variation based on UGMTS.

5.1 UGMTS

The UGMTS framework involves three parts: 1) Building searchable index locally; 2) Query construction based on the search interests of data users; 3) Search processing in cloud.

5.1.1 Building Index

A data owner builds a searchable index I in local before outsourcing the data to cloud. The index I contains two index groups, that is $I = \{IC, IR\}$, where IC is used to select effective candidate documents, and IR is used to calculate the final relevance scores between queries and candidate documents. The details are as follows.

Firstly, the data owner creates an inverted index V for the dictionary W, the inverted index consists of a set of inverted lists and it is denoted as $V = \{vl(w_1), vl(w_2), .., vl(w_n)\}$. Where $vl(w_i)$ is the inverted list of keyword w_i and represents the top $c * k$ documents of keyword w_i (where c is a positive integer).

Note that, we have adopted the champion lists to our scheme as each inverted list only stores the top $c * k$ documents of each corresponding keyword, which can improve query efficiency and save storage space, but it may also result in lower query accuracy. However, data owners can control the side-effect by adjusting the value of c (in this paper, c is set to 1, in performance analysis we show that c has limited impact on query accuracy).

Secondly, the data owner divides dictionary W into multiple groups as $WG = \{WG_1, WG_2, ..., WG_b\}$, where each group only contains d keywords. The data owner also finds the top $c * k$ documents of each word group based on the inverted index V, denoted as $VG = \{VG_1, VG_2, .., VG_b\}$, where VG_i is the top $c * k$ documents of word group WG_i. To formulate this problem, we give the definition of *top-k documents of word group*:

***Definition* 3.** *(Top-k Documents of Word Group) Giving a word group $\{w_1, w_2, ..., w_d\}$, the top-k documents of this word group equal to $U(vl(w_1) \cup vl(w_2) \cup ... \cup vl(w_d))$, where function $U()$ is used to remove duplicated documents.*

Thirdly, the data owner builds a keyword balanced binary tree [16] as index for each keyword group from the corresponding top-ck documents (e.g. IC_i is the index of keyword group WG_i, and it is built from VG_i). The indexes combine as $IC = \{IC_1, IC_2, ..., IC_b\}$, N_i represents a node in index IC_i and it has form as $< fid, l_c, r_c, val >$, where l_c and r_c are children node of N_i, and val is a data vector with d dimensions. If N_i is a leaf node, then fid is its corresponding document identifier and val stores the TF values of keyword group WG_i (e.g., the $val[j]$ equals to the TF value of keyword $WG_{i,j}$ in document D_{fid}, where $WG_{i,j}$ is the j-th keyword in keyword group WG_i). Otherwise, if node N_i is an intermediate node, then the fid is empty and the val is computed as below:

$$val[j] = max\{l_c.val[j], r_c.val[j]\}, j = 1, 2, ..., d \tag{4}$$

Where $l_c.val$ and $r_c.val$ are the stored vectors of the left and right children of N_i, respectively.

Finally, the data owner constructs another index group IR for the document collection D, where $IR = \{IR_1, IR_2, ..., IR_m\}$. We use IR_{fid} to represent an index in IR. The index IR_{fid} is built based on document D_{fid}, and

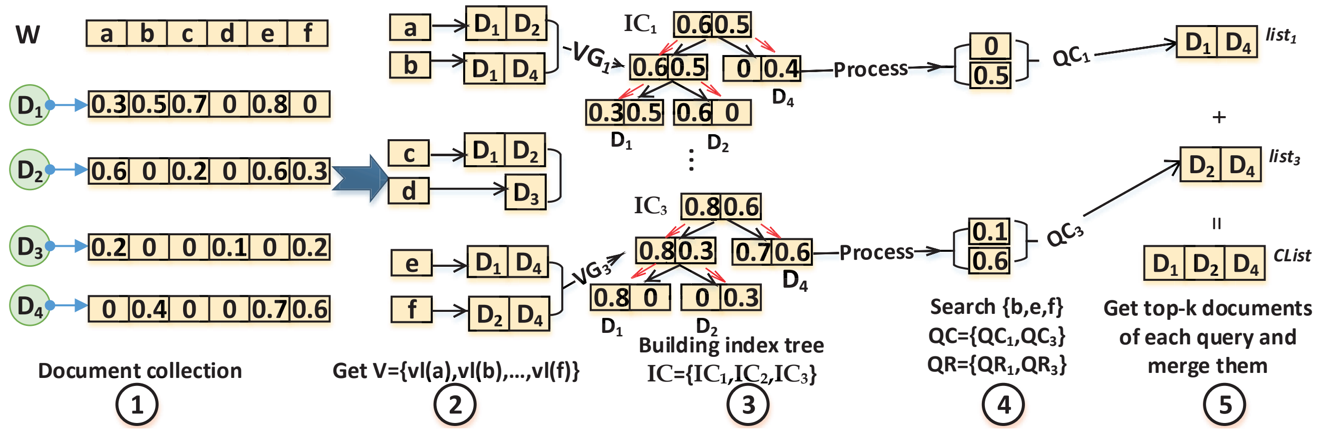


Fig. 5. This is an example of UGMTS. 1) The document collection D contains four documents $\{D_1, D_2, D_3, D_4\}$, the dictionary W has six keywords $\{a, b, c, d, e, f\}$. 2) The data owner builds an inverted list for each keyword and divides W into three groups where each group contains two words, then merges the inverted index V into three groups. 3) The data owner builds a searchable index for each keyword group. 4) The data user submits the search keywords $\{b, e, f\}$ which only contains the keywords in WG_1 and WG_3, so we build query groups $QC = \{QC_1, QC_3\}$ and $QR = \{QR_1, QR_3\}$, then QC and QR were send to cloud server. 5) The cloud server searches the top-2 documents on the index IC_1 for query QC_1 and searches the top-2 documents of QC_3 on the index IC_3, it gets results $list_1 = \{D_1, D_4\}$ and $list_3 = \{D_2, D_4\}$. Then the cloud server merges all results into collection $CList$ as candidate documents, where $CList = \{D_1, D_2, D_4\}$. Finally, the cloud server uses Equation 6 to calculate the final scores between the query group QR and all the documents in $CList$, and returns k documents with highest relevance scores to the data user as results.

it can be denoted as $IR_{fid} =< fid, val_1, val_2, ..., val_b >$, where val_i is a vector with length d and $val_i[j] = TF(WG_{i,j}, D_{fid})$ $(j = 1, ..., d)$.

5.1.2 Query Construction

When the data user wants to search with keyword set W_q, s/he generates query group Q. The query group is represented as $Q = \{QC, QR\}$, where QC is used to search on index group IC and QR will be processed in IR.

Query group QC is denoted as $\{QC_1, QC_2, ..., QC_b\}$. QC_i represents a query in QC and its a query vector with length d. The j-th dimension of query QC_i corresponds to keyword $WG_{i,j}$. If $WG_{i,j}$ exists in the keyword set W_q, the value of $QC_i[j]$ is the IDF of keyword $WG_{i,j}$, otherwise it is 0. Note that when all dimensions of query QC_i are 0, the data user would remove QC_i from QC. The other query group QR is denoted as $QR = \{QR_1, QR_2, ..., QR_b\}$ and it is the same as QC in UGMTS. Finally, the data user submits query group Q to the cloud server.

5.1.3 Query Processing

The details of search procedure in cloud servers are shown in Algorithm 1. When the cloud server receives query Q. Firstly, it processes QC on index group IC to get the candidate documents $CList$. Note that, each query in QC is only processed on its corresponding index (e.g. the query QC_i only be processed on index IC_i). The relevance scores between QC_i and the nodes of IC_i are calculated by Equation 5.

$$Score(QC_i, N_i) = (QC_i) \cdot (N_i.val) \tag{5}$$

Secondly, the cloud server uses Equation 6 to calculate the final relevance scores between query group QR and the documents in $CList$ on the index group IR, and returns k documents with the highest scores to the data user as results. An example is shown in Fig. 5.

$$Score(QR, IR_i) = \sum_{QR_j \in QR} (QR_j) \cdot (IR_i.val_j) \tag{6}$$

Algorithm 1 Search Process of UGMTS

Require: The query Q, the searchable index I;
Ensure: Return k documents with highest scores to the data user;

```
1: function SEARCH($Q$, $I$, $k$)
2:     for query $QC_i$ in query group $QC$ do
3:         FINDTOPK($QC_i$, root of $IC_i$, 0, $k$)
4:         Merge top-$k$ documents $list_i$ of $QC_i$ into $CList$
5:     end for
6:     for document $D_i$ in $CList$ do
7:         if Score($QR$, $IR_i$) > $k$-th score in $\widehat{Re}$ then
8:             Insert $i$ into $\widehat{Re}$
9:         end if
10:    end for
11:    return top-$k$ documents of $\widehat{Re}$
12: end function
13:
14: function FINDTOPK($QC_i$, $node$, $sco$, $k$)
15:    if sco < $k$-th score in $list_i$ then
16:        return
17:    end if
18:    if $node$ is leaf node then
19:        Insert the $fid$ of $node$ into $list_i$.
20:    else
21:        leftScore = Score($QC_i$, $node.l_c$)
22:        rightScore = Score($QC_i$, $node.r_c$)
23:        if leftScore > rightScore then
24:            FINDTOPK($QC_i$, $node.l_c$, $leftScore$, $k$)
25:            FINDTOPK($QC_i$, $node.r_c$, $rightScore$, $k$)
26:        else
27:            FINDTOPK($QC_i$, $node.r_c$, $rightScore$, $k$)
28:            FINDTOPK($QC_i$, $node.l_c$, $leftScore$, $k$)
29:        end if
30:    end if
31: end function
```

5.2 EGMTS

The UGMTS is efficient but less effective in privacy preservation for both data owners and data users. Therefore, for protecting the real value of indexes and queries, we add some phantom terms and random values into them to disturb the real relevance scores, and adopt the secure kNN algorithm [32] to encrypt them. The framework of the encrypted group multi-keyword top-k search scheme (EGMTS) is as follows:

Setup(d, u, r): The data owner generates two secret keys sk_1 and sk_2, where $sk_1 = \{S_1, M_1, M_2\}$ and $sk_2 = \{S_2, M_3, M_4\}$. S_1 contains a group of $(d+u+1)$-bit vectors and was denoted as $S_1 = \{S_1^1, S_1^2, ..., S_1^b\}$. S_2 contains $(b+1)$ randomly vectors, and the length of the first b vectors is $(d+r)$-bit, while the last vector occupies $(r+u+1)$-bit, thus we denote S_2 as $S_2 = \{S_2^1, S_2^2, ..., S_2^b, S_2^{b+1}\}$. Each bit of vectors in S_1 and S_2 is randomly set to 1 or 0. M_1 and M_2 are two groups of matrices, both of which contain b invertible matrices that are $(d+u+1) \times (d+u+1)$. While M_3 and M_4 both contain b invertible matrices with $(d+r) \times (d+r)$ and one $(r+u+1) \times (r+u+1)$ invertible matrices. When the secret keys are constructed, the data owner shares them with authorized data users.

BuildIndex(D, W, d): The method of building index I (where $I = \{IC, IR\}$) follows the same procedure as in UGMTS. Except that, some phantom terms are added into all the data vectors of index I, the details are:

1. *Magnify the Values of IC*: In UGMTS, each index of the index group IC is a keyword balanced binary tree, where the values of data vector of intermediate node are the max value from its children. But such relation may introduce security concerns as the cloud server can build more linear equations to calculate the plaintext information of indexes. To hide the relation, we magnify the value of data vectors by adding random numbers, such that Equation 4 is changed into $val[j] = max\{l_c.val[j], r_c.val[j]\} + |rand()|\%max\{l_c.val[j], r_c.val[j]\}$.
2. *Extending IC*: The dimension of each data vector in index group IC is extended from d to $d+u+1$, where u is the number of phantom terms. In addition, the values of phantom terms are randomly set to 0 or 1, and the $(d+u+1)$-th dimension of all these data vectors are set to 1.
3. *Extending IR*: Each vector in the index group IR is extended from d to $(d+r)$, and the values of extended r dimensions are the same within one index, but varies for different indexes. Data owner also adds a vector of length $(r+u+1)$ to each index, where u is the number of phantom terms, and the values of extended r dimensions are the same as other vectors in the same index. Note that, both the values of extended dimensions and phantom terms are set to 0 or 1, and all the $(r+u+1)$-th dimension of added vectors are set to 1.

EncryptIndex(IC, IR, sk_1, sk_2): The index groups IC and IR are encrypted before outsourcing. We use N_i to denote a node in index IC_i and NV_i to represent the stored data vector. Furthermore, we use $S_{1,i}$ to denote the i-th vector in S_1. Firstly, the data owner splits vector NV_i into two random vectors $\{NV_i^{'}, NV_i^{''}\}$ based on the value of $S_{1,i}$. Specifically, if $S_{1,i}[j]$ is 0, $NV_i^{'}[j]$ and $NV_i^{''}[j]$ are the same as $NV_i[j]$; if $S_{1,i}[j]$ is 1, $NV_i^{'}[j]$ and $NV_i^{''}[j]$ are set with two random numbers, but their summation equals to $NV_i[j]$.

After splitting process is complete, node N_i stores two encrypted vectors $\{M_{1,i}^T NV_i^{'}, M_{2,i}^T NV_i^{''}\}$, where $M_{1,i}$ and $M_{2,i}$ represent the i-th matrices in the matrix groups M_1 and M_2, respectively. The encryption form of index group IC is denoted as:

$$\begin{aligned}\widetilde{IC} &= \{M_1^T IC^{'}, M_2^T IC^{''}\} \\ &= \{\{M_{1,1}^T IC_1^{'}, M_{2,1}^T IC_1^{''}\}, ..., \{M_{1,b}^T IC_b^{'}, M_{2,b}^T IC_b^{''}\}\}\end{aligned}$$

The data owner also encrypts IR with secret key sk_2, where the encryption method is the same as encrypting IC. We use $\widetilde{IR}$ to represent the encrypted IR. Finally, the data owner outsources $\widetilde{IC}$, $\widetilde{IR}$ and the encrypted document collection C to cloud.

CreateQuery(W_q, key). The method of generating query groups QC and QR is similar to the UGMTS. However, the query vectors in QC and QR need to be extended, and the details are:

1. *Extending QC*: Each query in QC is independent when processing the search request, hence phantom terms are added to them all. Thus, the query vectors in QC are extended from d to $d+u+1$ dimensions, and the phantom terms are stored in the first u dimensions of the extension. The values of phantom terms are set to random numbers $\xi_{i,j}$ and the $(d+u+1)$-th dimension is set to another random number λ_i (where $i = 1, ..., b$ and $j = 1, ..., u$). Besides, the first $d+u$ dimensions of each vector are multiplied by a random positive number γ.
2. *Extending QR*: Before extending QR, a phantom query QR_{b+1} which contains a vector of $r+u+1$ dimensions was added to QR by data users. Moreover, to improve the query accuracy such phantom terms are only added into this phantom query. When calculating final relevance score, the cloud servers need to treat all the queries in QR as a whole, thus we add a number of r dummy keywords into each query and restrict their summation to be zero, which means that the stored vectors of the first b queries are extended to $d+r$, and the values of extended dimensions satisfy the requirement such as $QR_{b+1}[j] + \sum_{i=1}^{b} QR_i[d+j] = 0$ (where $j = 1, ..., r$). Note that, the above restriction is used to prevent the cloud servers from learning the final relevance scores between candidate documents and any single query, if the cloud server processes QR as a whole, it will get the real score, since the summation of all dummy keywords is zero. Otherwise, the final score will be the real score plus the score of partial dummy keywords. In addition, the $(r+j)$-th dimension of query vector QR_{b+1} is set to random number ξ_j (where $j = 1, ..., u$) and the $(r+u+1)$-th dimension

is set to a random positive number λ. Finally, all the query vectors in the query group QR are scaled by a random positive number γ except the last dimension.

EncryptQuery(QC, QR, key**)**. After query groups QC and QR are generated, the data user encrypts them with secret keys sk_1 and sk_2, respectively. Firstly, the data owner splits query QC_i into two random vectors $\{QC_i^{'}, QC_i^{''}\}$according to the value of vector $S_{1,i}$. If $S_{1,i}[j]$ is 0, the sum of $QC_i^{'}[j]$ and $QC_i^{''}[j]$ equals to $QC_i[j]$, otherwise $QC_i^{'}[j]$ and $QC_i^{''}[j]$ are the same as $QC_i[j]$ (where $j = 1, ..., d + u + 1$). Finally the query group QC is encrypted as below:

$$\begin{aligned}\widetilde{QC} &= \{M_1^{-1}QC^{'}, M_2^{-1}QC^{''}\} \\ &= \{\{M_{1,i}^{-1}QC_i^{'}, M_{2,i}^{-1}QC_i^{''}\} \mid QC_i \in QC\}\end{aligned}$$

The data user also uses secret key sk_2 to encrypt QR with the same method applied to QC. In the end, the data user submits $< \widetilde{QC}, \widetilde{QR} >$ to the cloud server as trapdoor where $\widetilde{QR}$ denotes the encrypted QR.

Query($\widetilde{QC}, \widetilde{QR}, \widetilde{IC}, \widetilde{IR}, k$**)**. The query processing method over the encrypted index groups $\widetilde{IC}$ and $\widetilde{IR}$ is similar to UGMTS, except the relevance scores between the nodes of index $\widetilde{IC}_i$ and query $\widetilde{QC}_i$ are calculated by Equation 7, and the score between the query group $\widetilde{QR}$ and index $\widetilde{IR}_\rho$ is calculated by Equation 8.

$$\begin{aligned}&Score(\widetilde{QC}_i, \widetilde{N}_i) \\ &= \{M_{1,i}^T NV_i', M_{2,i}^T NV_i''\} \cdot \{M_{1,i}^{-1}QC_i^{'}, M_{2,i}^{-1}QC_i^{''}\} \\ &= \gamma(Score(QC_i, N_i) + \sum_{j=1}^{u} \xi_{i,j}) + \lambda_i\end{aligned} \tag{7}$$

Where $\widetilde{N}_i$ represents a node in index $\widetilde{IC}_i$, $M_{1,i}^T NV_i'$ and $M_{2,i}^T NV_i''$ are the stored data vectors of $\widetilde{N}_i$.

$$\begin{aligned}&Score(\widetilde{QR}, \widetilde{IR}_\rho) \\ &= \sum_{\widetilde{QR}_i \in \widetilde{QR}} (\widetilde{IR}_\rho.val_i^{'}) \cdot (\widetilde{QR}_i^{'}) + (\widetilde{IR}_\rho.val_i^{''}) \cdot (\widetilde{QR}_i^{''}) \\ &= \gamma(Score(QR, IR_\rho) + \sum_{j=1}^{u} \xi_j) + \lambda\end{aligned} \tag{8}$$

Where $\widetilde{IR}_\rho$ represents an index in $\widetilde{IR}$, $\widetilde{IR}_\rho.val_i'$ and $\widetilde{IR}_\rho.val_i''$ are the stored query vectors in $\widetilde{IR}_\rho$.

5.3 Security Analysis

In this paper, we do not discuss the security of document collection, because it is encrypted by the data owner before outsourcing to the cloud server, and the encryption method could be any traditional encryption method that is suitable for the concern of data owners. Hence, we assume the encryption methods are secure and can guarantee strong data privacy. Next, we analyze the security of our scheme in known ciphertext model and known background model, respectively.

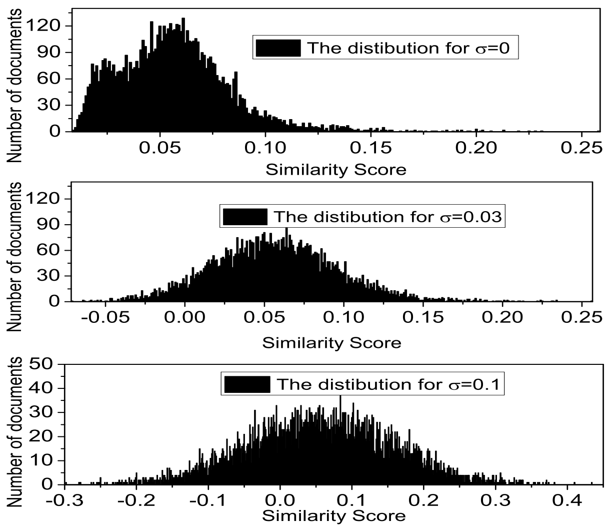


Fig. 6. The distribution of similarity score for keyword "network" with different values of σ.

5.3.1 *In Known Ciphertext Model*

Adversaries can calculate the real value of indexes and queries by establishing liner equations from the exposed ciphertext [15]. Assume $\widetilde{IC}_i$ represents an index in the index group $\widetilde{IC}$, and it is encrypted by the secure kNN algorithm with secret key sk_1, where each data vector is randomly split into two different vectors. The number of equations that established from the ciphertext of this index is $4\beta m(d+u+1)$, where $0 \le \beta \le 1$. But index $\widetilde{IC}_i$ contains $2\beta m$ nodes and $2(d+u+1)$ unknown numbers in each node, there also have $2(d+u+1)^2$ unknown numbers in matrices $M_{1,i}$ and $M_{2,i}$. It is obvious that the size of unknowns numbers is more than the known equations. Similarly, for the index group $\widetilde{IR}$ and trapdoor, the number of unknown numbers is also more than the known equations. Hence, adversaries have no sufficient equations to calculate the plaintext of indexes and trapdoors without secret key sk_1 and sk_2.

According to Yao et al. [35], the secure kNN algorithm is not secure against the chosen-plaintext attack. But next we prove our EGMTS is secure.

Proof: Yao et al. use the known plaintext-ciphertext pair of queries to construct linear equations to calculate the values of index, but in our scheme, the relevance score learned by the cloud server is

$$Score(\widetilde{QC}_i, \widetilde{N}_i) = \gamma(Score(QC_i, N_i) + \sum_{j=1}^{u} \xi_{i,j}) + \lambda_i$$

where $Score(QC_i, N_i)$ is real score, but it is disturbed by $2 + u$ random numbers (two random numbers γ and λ_i, u random numbers $\xi_{i,j}$), which means that even for two identical queries, the relevance scores are different. Besides, each linear equation may introduce $2+u$ unknown random numbers, therefore the unknowns in equations are always more than the number of linear equations, so adversaries cannot calculate the real value of index, and also cannot infer the real value of secret key. □

In summary, the EGMTS is strong enough to protect the security of index and query in known ciphertext model, and it is more secure than the secure kNN algorithm.

5.3.2 *In Known Background Model*

In the known background model, cloud servers have more knowledge about the stored data, such as the normalized TF distribution of some keywords, therefore, the cloud server can identify these keywords by comparing the normalized TF distributions [14], [15], [16], [30], [33]. In addition, cloud server may learn the search interests of data users by linking queries and exploiting access pattern.

Keyword Privacy: In EGMTS, keyword privacy can be guaranteed by inserting random numbers $\xi_{i,j}$ and ξ_j into queries to obscure the normalized TF distributions. Moreover, random numbers $\xi_{i,j}$ and ξ_j follow the uniform distribution $U(\mu' - \delta, \mu' + \delta)$. According to the central limit theorem, the $\sum_{j=1}^{\omega} \xi_{i,j}$ and $\sum_{j=1}^{\omega} \xi_j$ follow the normal distribution $N(\mu, \sigma^2)$ (where $2\omega = u$, $\mu = u\mu'/2$ and $\sigma^2 = u\delta^2/6$). As shown in Fig. 6, the bigger is the value of σ, the higher is the level of interference, but the lower is the query accuracy. Therefore, data users can balance the trade off between query accuracy and keyword privacy by adjusting the value of σ.

Query Unlinkability: In EGMTS, even though each query only contains one keyword, it is also represented as a fixed-length vector. Hence, the cloud server cannot determine which keywords the data user wants to search. In addition, each query vector is randomly split into two different vectors which are encrypted before processing, and the relevance scores are also disturbed by inserted random numbers. Hence, the cloud server cannot distinguish even the same search request just rely on ciphertext and relevance scores. Moreover, according to [14], data users can control the level of query unlinkability by adjusting the value of phantom terms.

However, the cloud server can link two queries by comparing and analyzing the access pattern and visiting paths on the index. Even though the relevance scores and visiting paths are obscured by inserting random numbers, the accuracy of queries usually get reduced with the increasing interference, which is impractical. To better ensure the system availability, the accuracy of queries cannot be too low. But with the increase of query accuracy, the access pattern and visiting paths of two identical queries are becoming more similar (e.g., in order to guarantee the correct ratio is above 80%, two identical queries must share at least 60% common results and visiting nodes). Hence, the EGMTS and EMTS [14], MRSE_II [15] and EDMRS [16] cannot perfectly protect the query unlinkability.

6 The scheme of RGMTS

In Section 4 we have introduced the random traversal algorithm which can change the visiting paths and search results of two identical queries by using different keys. In last section we also have introduced that EGMTS has weakness in query unlinkability protection, since the cloud server can link two queries by comparing and analyzing their visiting paths and search results. In this section, we propose a random group multi-keyword top-k search scheme (RGMTS) which absorbs the advantages of both RTRA and EGMTS, and provides more data security than EGMTS.

6.1 Building RGMTS Index

First, the data owner enlarges the document collection D to DG^x and assigns a random switch to each document, where the method is the same as RTRA. For example, when both E and L are set to 2, the document collection $D = \{D_1, D_2, D_3, D_4\}$ is extended to $DG^x = \{D_1^1, D_2^1, D_4^2, D_3^2, D_2^2, D_1^2, D_3^1, D_4^1\}$.

Then, similar to the index construction in GMTS, the data owner divides all the keywords in dictionary W into several keyword groups and finds the top-ck documents of each word group. But in RGMTS, all the top-ck document groups are further extended, such as that VG_i is extended to VC_i^x where VG_i is the top-ck documents of keyword group WG_i, and VC_i^x is a subset of DG^x which contains all the copies of documents belong to VG_i. Note that, the documents that belong to VG_i^x keep the same order as DG^x.

e.g., keyword group WG_1 contains two keywords $\{a, b\}$ and its top-ck documents are $VG_1 = \{D_1, D_2, D_4\}$, after VG_1 is extended to VG_1^x, the data owner gets $VG_1^x = \{D_1^1, D_2^1, D_4^2, D_2^2, D_1^2, D_4^1\}$, where $VG_1^x \subseteq DG^x$.

The data owner uses the extended top-ck documents to build a searchable index for each keyword group, by using the method which has been applied to EGMTS. For instance, the data user uses VG_i^x instead of VG_i to build a searchable index for keyword group WG_i. Suppose we use $< fid, l_c, r_c, val >$ to represent one node of these searchable indexes, where fid is the document identifier, l_c and r_c are the left and right child, respectively, val is a data vector of e (where $e = d + u + r + 1$). We also specified that the first d dimensions of the vector are the TF of its corresponding keywords, the $(d + j)$-th (where $j = 1, ..., u$) dimension stores phantom terms, the $(d + u + j)$-th dimension stores the switch of this node (where $j = 1, ..., r$), and the e-th dimension is set to 1. The data owner also builds an index group IR for document collection DG^x, where the switch is stored in the added data vector. Finally, the data owner encrypts collection DG^x, index groups IC and IR, and sends them to cloud.

6.2 Search Process of RGMTS

When the data user wants to search with keyword set W_q, s/he constructs two query groups QC and QR using the method which is similar to EGMTS, except that:

1. The query vector in query group QC is extended from $(d + u + 1)$ to e;
2. Each query of QC is assigned a random key, and these keys are stored in the $(d + u + 1)$-th dimension to $(d + u + r)$-th dimension of each vector.
3. The data user assigns a random key to the phantom query in the query group QR.

The data user encrypts QC and QR, and sends them to cloud as trapdoor T. When processing the query $\widetilde{QC_i}$, which represents a query of $\widetilde{QC}$, the cloud server calculates the relevance scores between $\widetilde{QC_i}$ and the nodes of index $\widetilde{IC_i}$ from the root to the leaf, but only when the score of one node is larger than zero, its children nodes will be traversed. After that, the cloud server merges all the results into $CList$ as candidate documents and calculates the relevance scores between the documents in $CList$ and the query group

$\widetilde{QR}$. Finally, the cloud server returns k documents with the highest relevance scores to the data user as search results.

6.3 Security Analysis

The RGMTS takes the advantage of RTRA which make sure that if the data user submits two identical queries with different keys, our search procedure in the cloud server must have different visiting paths and results, and in the meantime it maintains the accuracy of queries unchanged. In addition, it is easy to conclude that the probability of getting the same query results and visiting paths of two identical queries is less than $1/E^L$, and the expectation of the number of common documents between the two search results is less than $|\widehat{Re}|/E$. Therefore, we can control the level of query unlinkability by adjusting the value of E and L, and without sacrificing the correct ratio.

In order to completely hide access pattern and visiting paths, which requires the probability of getting the same results and visiting paths of two identical queries must be less than or equal to that the probability of two different queries. Therefore, the value of E would be very large, but with the increasing of value E, the index space is also becoming larger (even though we can decrease the storage space by only store the top-ck documents), thus data owners have to balance the trade-off between data security and storage space by adjusting the value of E.

In summary, the RGMTS trades space for data security, which can better protect the query unlinkability and access pattern than most existing works (such as [14], [15], [16]).

7 PERFORMANCE ANALYSIS

In this section, we first describe the experimental setup and scenarios, then we analyze the performance of our schemes from two aspects: 1) the precision and privacy; 2) the efficiency of index construction, trapdoor generation and query processing. As one reference point, for query efficiency, we compare the time cost of our solution with the approach EDMRS as proposed in [16], which represents the latest research finding with high query efficiency.

7.1 System Implementation

The overview of our system is shown in Fig. 1, the main functions of three module are briefly summarized as follow: 1) the data owner encrypts raw collection D to get encrypted version C, and builds searchable index I_e based on C; 2) the data user encrypts the query to construct trapdoor T, by using the key as shared by the data owner, and get the encrypted query results from the cloud server; 3) the cloud server stores the outsourced C and I_e from the data owner, it traverses the index to process encrypted queries, and returns those documents with top-k highest scores. To test the performance of our schemes, we implement our system based on GMTS and RGMTS. In particular, the former two modules are implemented with C language and Python on a Windows 10 PC with Intel(R) Core(TM) i5-4590 CPU 3.30GHz and 4 Gigabyte memory. The cloud server module is implemented with C language on a Linux Server with Intel(R) Xeon(R) CPU E5620 Processors 2.40GHz and 24 Gigabyte memory. For convenience and fairness, we implement EDMRS and MRSE on the same programming languages and platform as that of GMTS and RGMTS.

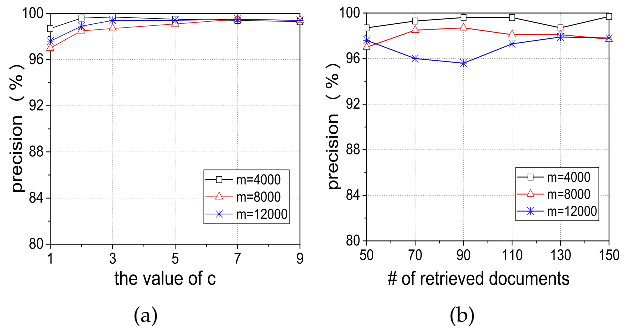


Fig. 7. The impact of c on precision, we set $n = 4000$. (a) For the different values of c with the same k, we set $k = 50$. (b) For the different values of k with the same c, we set $c = 1$.

Dataset: We randomly select 12000 emails as our collection D from a publicly available real-life data set: the Enron Email Data Set [36]. In addition, we randomly extract 12000 keywords from these emails as our dictionary with Python, where each keyword is processed by Porters stemming algorithm [37] which excludes the stop words.

7.2 Precision and Privacy

Without loss of generality, we adopt the definition of "precision" from [15], in which precision P_k is defined as $P_k = k'/k$, where k' is the number of real top-k documents in query results.

As described in Section 5.1.1, to decrease the size of indexes and improve the query efficiency, we adopt *champion lists* to our schemes, where each index only stores the top-ck documents of its corresponding keyword group. It is obvious that the previous methods may result in lower accuracy of queries. However, the data user in our scheme can control the query accuracy by tuning c. As shown in Fig. 7 that the larger value of c is, the higher is the precision we get. Fig. 7 also indicates that c has only limited impacts when its value beyond a certain point, that is because most of the top-k documents in a multi-keyword query appear in the union of search results for one single keyword query.

The works [14], [15], [16] are not designed to protect access pattern, and they only add random number ξ_j into index or queries to control the level of query unlinkability. However, as we know the adversaries can link two queries by comparing and analyzing the access pattern, thus these works cannot protect the query unlinkability perfectly. In our work, we not only adopt random numbers to control the query unlinkability, but we also proposed RTRA to hide the access pattern. Although the access pattern is hard to hide thoroughly, we can reduce its leakage and increase the difficulty of cloud servers to link two identical queries. One obvious observation is that if the number of common documents between the query results of two identical queries becomes smaller, then it will be more difficult for cloud servers to distinguish from these two queries. Hence, we define the level of query unlinkability as $d_i = 1 - k''/k$, where k'' is the number of common documents between the two query results. In RGMTS, we know that the number k'' for two identical queries decreases as the value of E increases. Therefore, as shown in Fig. 8(a), the level of query

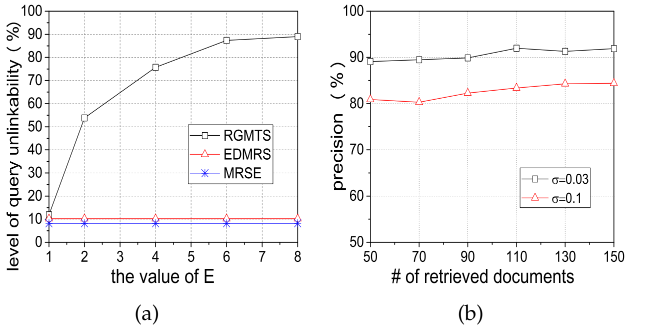


Fig. 8. We set $n = 4000$, $m = 4000$. (a) The level of query unlinkability with fixed value $\sigma = 0.03$ and different values of E. (b) The precision with different values of σ.

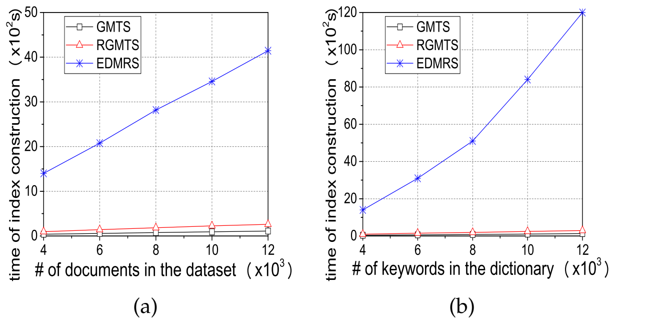


Fig. 9. Time cost of index construction. (a) for different sizes of the document collection D and fixed size of the dictionary W with $n = 4000$. (b) for different sizes of dictionary W and fixed size of D with $m = 4000$.

unlinkability of RGMTS is increased with larger E, but EDMRS [16] and MRSE [15] do not have such property. On the other hand, to protect the privacy of keyword in the query, random number ξ_j was added to each query, which can directly affect the query precision. Fig. 8(b) shows the influence of ξ_j on precision, where ξ_j follows the normal distribution $N(\mu, \sigma^2)$, and σ is the standard deviation.

7.3 Efficiency

7.3.1 Index construction

The procedure of index construction can be divided into two steps: 1) building a tree-based index group IC for all the keywords and constructing an index IR for document collection D; 2) encrypting all nodes in the indexes with secret keys sk_1 or sk_2.

As introduced in previous section, when building the index IC, each index only stores the top-ck documents of its corresponding keyword group. Therefore, the above operation only generates $O(\beta mb)$ nodes, where β is a decimal which is less than or equal to 2. On the other hand, our scheme generates $O(mb)$ nodes when building index IR. Overall, there are totally $O(\theta mb)$ nodes will be generated during the index construction, where $\theta = 1 + \beta$.

Node encryption needs a splitting process and two multiplications of $e \times e$ matrix, where e is the length of vector in each node, which equals to $(d + u + 1)$ in GTMS and $(d + u + r + 1)$ in RGMTS (we ignore the different length of vectors in IC and IR). The splitting process takes $O(d)$ time and the two multiplications takes $O(e^2)$ time. Hence, we can conclude that the time complexity of index construction is approximately equal to $O(\theta mbe^2)$. Note that, in this paper, we compare our schemes with EDMRS, which is more efficient than MRSE [15] and other methods. The time complexity of EDMRS is $O(n^2 m)$. It is obvious that our schemes are mainly influenced by e, but EDMRS is proportional to the square of n, where $n = |W|$. On the other hand, the index of EDMRS is encrypted by two $n \times n$ matrices, which is time-consuming. But the indexes of our schemes are encrypted by several $e \times e$ matrices, where e is smaller than n. So, our schemes usually spend less time in encrypting the indexes. As shown in Fig. 9, our schemes are more efficient than EDMRS in index construction. Table 2 shows the storage overhead of our indexes ($m = 4000$, $E = 2$ and $c = 1$).

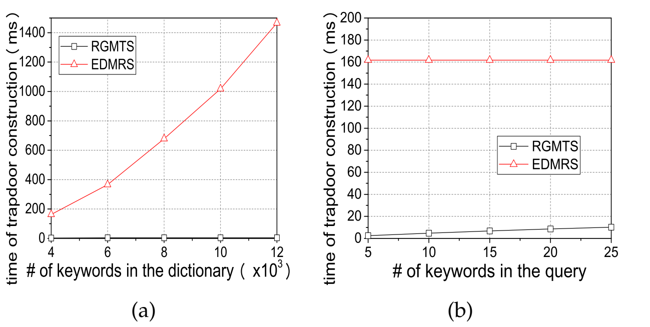


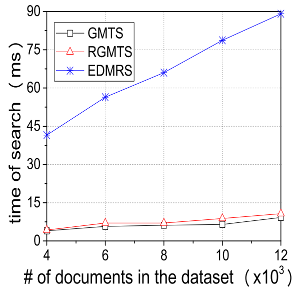
Fig. 10. Time cost of trapdoor construction. (a) for different sizes of dictionary W and fixed size of query with $t = 10$. (b) for different sizes of query and fixed size of dictionary W with $n = 4000$.

TABLE 2
Size of index

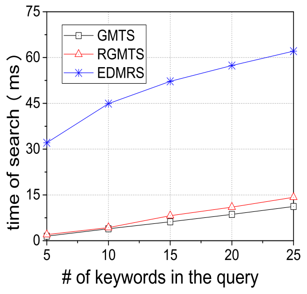
Size of dictionary	2000	4000	6000	8000
GMTS (MB)	159	314	473	574
RGMTS (MB)	345	682	1027	1232

7.3.2 Create Trapdoor

In EDMRS, no matter how many keywords are contained in the query W_q, the length of trapdoor is always equal to the size of dictionary, where the trapdoor is a vector. However, in most of the time, people are likely to search just with five keywords or less [14], [38]. Therefore, most dimensions of the trapdoor are equal to 0, which wastes the computing resources greatly. In our schemes, the trapdoor is divided into b parts, like the dictionary W, and each part is called a query which is a vector with length d. If all dimensions of a query are equal to 0, we remove it from the trapdoor. For example, assume our dictionary W contains $n = 8000$ keywords, and we divide it into $b = 100$ groups where each group contains $d = 80$ keywords. If the size of W_q is 5, then the generated trapdoor at most contains 5 queries, where each query includes one 80-dimensional vector. Therefore, the total length of our trapdoor is 400 at most, but the length of the trapdoor in EDMRS is 8000. Hence, in our schemes, the data user can take less time in trapdoor encryption. The time complexity of trapdoor construction in both GMTS and RGMTS are $O(te^2)$, where t is the number of queries in trapdoor. In the worst case, t equals to the size of W_q. Obviously, the time complexity is independent of the size of dictionary, and it is only affected by the size of vectors in queries and the size of trapdoor. In Fig. 10, we can

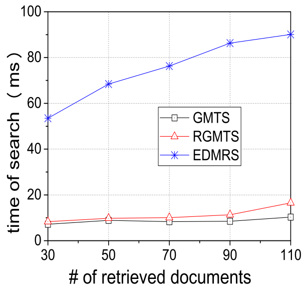


(a)

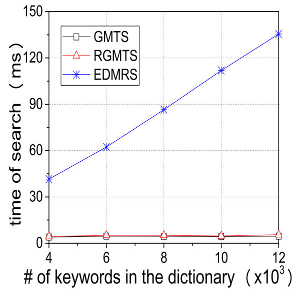


(b)

Fig. 11. Time cost of search. (a) for fixed size in query W_q ($t = 10$) and fixed size of dictionary ($n = 4000$) in different sizes of document collection. (b) for different sizes t in query W_q with fixed size of dictionary ($n = 4000$) and fixed size of document collection ($m = 4000$).



(a)



(b)

Fig. 12. Time cost of search. (a) for fixed n and m with different values of k (where k is the number of documents that the data user wants to retrieve, and $n = 4000$, $m = 8000$, $t = 10$). (b) the time cost of search with different sizes of the dictionary W, we set the size of the document collection D as $m = 4000$, and $t = 10$.

observe that our proposed scheme significantly outperforms the EDMRS in trapdoor construction.

7.3.3 Search

We improve query efficiency in three ways: 1) we build a searchable index for each keyword group instead of the whole dictionary; 2) each index only stores the top-ck documents of its corresponding keyword group; 3) every index is structured as a keyword balanced binary tree.

As mentioned above, the data user divides the original query into several queries and only sends non-empty queries to the cloud server. Therefore, with the first method, the cloud server does not need to search the indexes of all keyword groups. On the other side, the number of nodes in indexes was decreased with the second method, which avoid excessive search on extraneous nodes. Besides, with the third method, when we calculate the relevance scores between any node and queries, if the score of one node is less than the minimum score in $CList$, then its children nodes will not be traversed, thus many nodes could be pruned during our traversal process. With these methods, the overall computational cost is greatly reduced in our search procedure, and in the meantime we can guarantee the query privacy. Because the number of keywords in a query can be ranged from 1 to d, the cloud server cannot identify which keywords the data user wants to search.

We compare the query efficiency of our methods with EDMRS under different parameter settings. In particular, we study m (dataset size), t (query size), n (dictionary size) and the effect of k (parameter k in our top-k query) on real datasets. All results in Fig. 11 and Fig. 12 demonstrate that our methods are much more efficient in search time. In particular, Fig. 12(a) shows that the query time of each method increases with k since they all need more time to process the data. Similarly, both figures in Fig. 11 show that the query time of each algorithm increases with m and t, respectively. On the other hand, the time cost of query in our methods is independent of the dictionary size. So, as shown in Fig. 12(b), the efficiency of query in EDMRS drops sharply with the increased size n of dictionary, but our methods still maintain high efficiency.

8 Conclusion

In this paper, we focus on improving the efficiency and the security of multi-keyword top-k similarity search over encrypted data. At first, we propose the random traversal algorithm which can achieve that for two identical queries with different keys, the cloud server traverses different paths on the index, and the data user receives different results but with the same high level of query accuracies in the mean time. Then, in order to improve the search efficiency, we design the group multi-keyword top-k search scheme, which divides the dictionary into multiple groups and only needs to store the top-ck documents of each word group when building index. Next, to protect the query unlinkability, we apply the random traversal algorithm to get the RGMTS, which can increase the difficulty of cloud servers to conduct linkage attacks on two identical queries, and we can also tune the value of E to make the level of query unlinkability flexible for data owners. Finally, the experimental results show that our methods are more efficient and more secure than the state-of-the-art methods.

Acknowledgment

This work is supported by the NSFC under grant 61472148. The authors would like to thank the editor and reviewers for their insightful comments.

References

[1] J. Tang, Y. Cui, Q. Li, K. Ren, J. Liu, and R. Buyya, "Ensuring security and privacy preservation for cloud data services," *ACM Computing Surveys*, 2016.

[2] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "A view of cloud computing," *Communications of the ACM*, vol. 53, no. 4, pp. 50–58, 2010.

[3] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky, "Searchable symmetric encryption: Improved definitions and efficient constructions," in *Proceedings of the 13th ACM Conference on Computer and Communications Security*. ACM, 2006, pp. 79–88.

[4] D. Boneh, G. Di Crescenzo, R. Ostrovsky, and G. Persiano, "Public key encryption with keyword search," in *Advances in Cryptology-Eurocrypt 2004*. Springer, 2004, pp. 506–522.

[5] Z. Ying, H. Li, J. Ma, J. Zhang, and J. Cui, "Adaptively secure ciphertext-policy attribute-based encryption with dynamic policy updating," *Sci China Inf Sci*, vol. 59, no. 4, pp. 042 701:1–16, 2016.

[6] D. X. Song, D. Wagner, and A. Perrig, "Practical techniques for searches on encrypted data," in *Security and Privacy, 2000. SP 2000. Proceedings. 2000 IEEE Symposium on*, 2000, pp. 44–55.

[7] E.-J. Goh *et al.*, "Secure indexes." *IACR Cryptology ePrint Archive*, vol. 2003, p. 216, 2003.

[8] Y.-C. Chang and M. Mitzenmacher, "Privacy preserving keyword searches on remote encrypted data," in *Applied Cryptography and Network Security*. Springer, 2005, pp. 442–455.

[9] Y. H. Hwang and P. J. Lee, "Public key encryption with conjunctive keyword search and its extension to a multi-user system," in *Pairing-Based Cryptography–Pairing*. Springer, 2007, pp. 2–22.

[10] P. Golle, J. Staddon, and B. Waters, "Secure conjunctive keyword search over encrypted data," in *Applied Cryptography and Network Security*. Springer, 2004, pp. 31–45.

[11] L. Ballard, S. Kamara, and F. Monrose, "Achieving efficient conjunctive keyword searches over encrypted data," in *Information and Communications Security*. Springer, 2005, pp. 414–426.

[12] D. Boneh and B. Waters, "Conjunctive, subset, and range queries on encrypted data," in *Theory of cryptography*. Springer, 2007, pp. 535–554.

[13] E. Shen, E. Shi, and B. Waters, "Predicate privacy in encryption systems," in *Theory of Cryptography*. Springer, 2009, pp. 457–473.

[14] W. Sun, B. Wang, N. Cao, M. Li, W. Lou, Y. T. Hou, and H. Li, "Privacy-preserving multi-keyword text search in the cloud supporting similarity-based ranking," in *Proceedings of the 8th ACM SIGSAC Symposium on Information*, ser. ASIA CCS '13. ACM, 2013, pp. 71–82.

[15] N. Cao, C. Wang, M. Li, K. Ren, and W. Lou, "Privacy-preserving multi-keyword ranked search over encrypted cloud data," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 1, pp. 222–233, 2014.

[16] Z. Xia, X. Wang, X. Sun, and Q. Wang, "A secure and dynamic multi-keyword ranked search scheme over encrypted cloud data," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 2, pp. 340–352, 2016.

[17] C. D. Manning, P. Raghavan, H. Schütze *et al.*, *Introduction to information retrieval*. Cambridge university press Cambridge, 2008, vol. 1, no. 1.

[18] S. Brin and L. Page, "The anatomy of a large-scale hypertextual web search engine," *Computer Networks and ISDN Systems*, vol. 30, no. 17, 1998.

[19] J. Baek, R. Safavi-Naini, and W. Susilo, "Public key encryption with keyword search revisited," in *Computational Science and Its Applications*. Springer, 2008, pp. 1249–1259.

[20] D. J. Park, K. Kim, and P. J. Lee, "Public key encryption with conjunctive field keyword search," in *Information security applications*. Springer, 2004, pp. 73–86.

[21] W. M. Liu, L. Wang, P. Cheng, K. Ren, S. Zhu, and M. Debbabi, "Pptp: Privacy-preserving traffic padding in web-based applications," *IEEE Transactions on Dependable and Secure Computing*, vol. 11, no. 6, Nov 2014.

[22] J. Li, Q. Wang, C. Wang, N. Cao, K. Ren, and W. Lou, "Fuzzy keyword search over encrypted data in cloud computing," in *INFOCOM, 2010 Proceedings IEEE*, 2010, pp. 1–5.

[23] C. Wang, N. Cao, J. Li, K. Ren, and W. Lou, "Secure ranked keyword search over encrypted cloud data," in *Distributed Computing Systems (ICDCS), IEEE 30th International Conference on*, 2010, pp. 253–262.

[24] C. Wang, K. Ren, S. Yu, and K. M. R. Urs, "Achieving usable and privacy-assured similarity search over outsourced cloud data," in *INFOCOM, 2012 Proceedings IEEE*, 2012, pp. 451–459.

[25] M. Chuah and W. Hu, "Privacy-aware bedtree based solution for fuzzy multi-keyword search over encrypted data," in *Distributed Computing Systems Workshops (ICDCSW), the 31st International Conference on*, 2011, pp. 273–281.

[26] B. Wang, S. Yu, W. Lou, and Y. T. Hou, "Privacy-preserving multi-keyword fuzzy search over encrypted data in the cloud," in *INFOCOM, 2014 Proceedings IEEE*, 2014, pp. 2112–2120.

[27] M. Kuzu, M. S. Islam, and M. Kantarcioglu, "Efficient similarity search over encrypted data," in *Data Engineering (ICDE), 2012 IEEE 28th International Conference on*, 2012, pp. 1156–1167.

[28] Q. Lv, W. Josephson, Z. Wang, M. Charikar, and K. Li, "Multi-probe lsh: Efficient indexing for high-dimensional similarity search," in *Proceedings of the 33rd International Conference on Very Large Data Bases*. VLDB Endowment, 2007, pp. 950–961.

[29] X. Yuan, H. Cui, X. Wang, and C. Wang, "Enabling privacy-assured similarity retrieval over millions of encrypted records," in *European Symposium on Research in Computer Security*. Springer, 2015, pp. 40–60.

[30] C. Wang, N. Cao, K. Ren, and W. Lou, "Enabling secure and efficient ranked keyword search over outsourced cloud data," *IEEE Transactions on Parallel and Distributed Systems*, vol. 23, no. 8, pp. 1467–1479, 2012.

[31] H. Li, Y. Yang, T. H. Luan, X. Liang, L. Zhou, and X. Shen, "Enabling fine-grained multi-keyword search supporting classified sub-dictionaries over encrypted cloud data," *IEEE Transactions on Dependable and Secure Computing*, vol. 13, no. 3, pp. 312–325, May-June 2016.

[32] W. K. Wong, D. W.-l. Cheung, B. Kao, and N. Mamoulis, "Secure knn computation on encrypted databases," in *Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data*. ACM, 2009, pp. 139–152.

[33] S. Zerr, D. Olmedilla, W. Nejdl, and W. Siberski, "Zerber+r: Top-k retrieval from a confidential index," in *Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology*. ACM, 2009, pp. 439–449.

[34] I. H. Witten, A. Moffat, and T. C. Bell, *Managing gigabytes: compressing and indexing documents and images*. Morgan Kaufmann, 1999.

[35] B. Yao, F. Li, and X. Xiao, "Secure nearest neighbor revisited," in *Data Engineering (ICDE), 2013 IEEE 29th International Conference on*, 2013, pp. 733–744.

[36] W. Cohen., "Enron email data set," 2015. [Online]. Available: https://www.cs.cmu.edu/∼./enron/

[37] M. F. Porter, "An algorithm for suffix stripping," *Program*, vol. 14, no. 3, pp. 130–137, 1980.

[38] "Keyword and search engines statistics," 2016. [Online]. Available: http://www.keyworddiscovery.com/keyword-stats.html

PLACE PHOTO HERE

Xiaofeng Ding is currently working as an associate professor in the School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, China. He received his PhD degree in Computer Science from Huazhong University of Science and Technology, Wuhan, China in 2009. His research interests mainly include data privacy and query processing, data encryption, graph databases and crowdsourcing.

PLACE PHOTO HERE

Peng Liu received his B.S. degree in Computer Science and Technology from Huazhong Agricultural University in 2014. He is currently a Master student at the School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, China. His research interests mainly include data encryption, top-k search and query processing.

PLACE PHOTO HERE

Hai Jin received the PhD degree in computer engineering in 1994 from Huazhong University of Science and Technology (HUST), China, where he is currently the Cheung Kong professor of the School of Computer Science and Technology. In 1996, he was awarded a German Academic Exchange Service Fellowship to visit the Technical University of Chemnitz in Germany. He worked at the University of Hong Kong between 1998 and 2000, and as a visiting scholar at the University of Southern California between 1999 and 2000. He was awarded the Distinguished Young Scholar Award from the National Science Foundation of China in 2001. He is the chief scientist of ChinaGrid, the largest grid computing project in China. He is the member of Grid Forum Steering Group (GFSG). He has coauthored 15 books and published more than 500 research papers. His research interests include distributed computing, computer architecture, virtualization technology, cluster computing and grid computing, big data privacy and security, crowdsourcing, network storage, and network security. He is a senior member of the IEEE and a member of the ACM.