IET Information Security

Research Article

Privacy preserving weighted similarity search scheme for encrypted data

ISSN 1751-8709
Received on 3rd October 2017
Revised 3rd May 2018
Accepted on 21st June 2018
E-First on 6th September 2018
doi: 10.1049/iet-ifs.2018.5187
www.ietdl.org

Cheng Guo[1,2], Pengxu Tian[1,2], Chin-Chen Chang[3] ✉
[1]School of Software Technology, Dalian University of Technology, Dalian 116620, People's Republic of China
[2]Key Laboratory for Ubiquitous Network and Service Software of Liaoning Province, Dalian 116620, People's Republic of China
[3]Department of Information Engineering and Computer Science, Feng Chia University, Taichung 40724, Taiwan
✉ *E-mail: alan3c@gmail.com*

Abstract: Cloud computing has become increasingly popular among individuals and enterprises because of the benefits it provides by outsourcing their data to cloud servers. However, the security of the outsourced data has become a major concern. For privacy concerns, searchable encryption, which supports searching over encrypted data, has been proposed and developed rapidly in secure Boolean search and similarity search. However, different users may have different requirements on their queries, which mean different weighted searches. This problem can be solved perfectly in the plaintext domain, but hard to be addressed over encrypted data. In this study, the authors use locality-sensitive hashing (LSH) and searchable symmetric encryption (SSE) to deal with a privacy preserving weighted similarity search. In the authors' scheme, data users can generate a search request and set the weight for each attribute according to their requirements. They treat the LSH values as keywords and mix them into the framework of SSE. They use homomorphic encryption to securely address the weight problem and return the top-k data without revealing any weight information of data users. They formally analysed the security strength of their scheme. Extensive experiments on actual datasets showed that their scheme is extremely effective and efficient.

1 Introduction

Cloud computing has achieved great success in recent years. As a new technology of computing, cloud computing possesses huge advantages, including great computing power, low cost, large scale, and quick deployment. So, individuals and enterprises prefer to outsource their data to cloud servers. However, the security of cloud computing has drawn considerable attention. As long as data are outsourced to a cloud server, the management and the ownership of the data are separated. The data owner loses direct control of the data. The cloud server provider may obtain the specific content of the data owners' outsourced data. Some sensitive information, such as e-mails, photos, or personal health records, can be revealed. Encrypting the data before outsourcing may be a good solution to this problem. However, traditional search algorithms only support searching operations in the plaintext domain. The data that have been encrypted lose the characteristic to be searched. Another solution to this problem is that, when a data user wants to perform a search operation for some data, all the data stored in the cloud must be downloaded and decrypted locally. Then, the data user can search no matter what he/she wants. Unfortunately, if a data user downloads all the data, it will cause a large amount of communication overhead. In addition, if the data user performs the search operation locally, the cloud server can only be treated as a storage mode instead of a model that has great power in computing. Hence, this solution will make the search more tedious, which is contradictory to the advantage of utilising cloud computing. Therefore, it is essential to develop a secure and efficient scheme for searching in the ciphertext domain.

Song *et al.* [1] first defined the conception of searchable encryption (SE) and proposed the scheme of it. The scheme is based on a cryptographic primitive that supports searching in the ciphertext domain. Subsequently, many SE schemes [2–15] have been proposed to support different search functions. However, all of them are focused on the exact keyword search, which means that the search results must contain a single keyword related to the search request.

With the rapid growth of data volume, there are more and more high-dimensional data to deal with. However, there rarely exists an exact result for a high-dimensional search request. And what the users always want are only the data that are similar to what they are searching for. Therefore, searching data that is similar to a high-dimensional search request will be more practical and common. The k-nearest neighbour (kNN) is a state-of-the-art algorithm that deals with the similarity search for high-dimensional data. Each data item can be treated as a point in high-dimensional space. A kNN problem consists of the k points that are closest to the query point. The distance can be evaluated by using the Euclidean distance or the Manhattan distance. However, even a simple kNN algorithm will entail a lot of computing overhead. We need to compute all the distances and then compare them. Hence, we use a hash function called locality-sensitive hashing (LSH) to decrease the time for similarity search. The traditional hash function can perform a quick search operation in a hash table by decreasing the collisions of the hash value. Different from the traditional hash function, LSH can leverage the collisions to measure the similarity. The main idea of LSH is that two similar data would be mapped into one bucket with a very high probability. And two dissimilar data would be mapped into one bucket with a very low probability. In other words, the LSH values of two similar data items would be the same. Then, we can compute the LSH values of the data to build a hash table. When performing a search operation, we compute the LSH value of the searched data item and match it in the hash table. The bucket whose identifier collides with the query would include data similar to the query. Some SE schemes [16–20] also employ LSH to solve similarity search problems over encrypted data.

However, there would be numerous data users in a practical cloud server. And the searched data item would contain different attributes. Different data users may have their own preference for some attributes, which we call it the weights of attributes. This kind of preference will influence the final result for a similarity search. Given a same searched data item from data A and data B, the cloud server would return the same result to them. However, if user A sets weights to his query in some attributes, the cloud server will not provide the previous result.

For example, in Fig. 1, the data user wants to securely search the nearest neighbour point of (29, 7000, 180). The cloud server will return D1 to the user by comparing the Euclidean distance. However, if the user considers that age is more important than the other attributes in his query, the cloud server should return D2 rather than D1 to him/her. This problem can be solved perfectly in the plaintext domain. The Euclidean distance or other methods can be used to evaluate the similarity between two vectors. However, it gets complicated when the data are encrypted. In traditional SSE schemes, the data owner generates a static, encrypted index and outsources it to the cloud server. Then, the cloud server searches on the index according to the trapdoor that is generated by the authenticated data user. The cloud server will return the encrypted files that address what the data user searched for. This process will be more complicated when performing a secure weighted similarity search. The similarity between data records or attributes will disappear after the data are encrypted. Furthermore, the indices are always static and difficult to update. However, there would be an infinite number of trapdoors generated by a searched data item by varying different weights. To successfully complete the match between the index and the trapdoor, they should be assigned the same weights. However, the cloud server cannot store every possible static index to meet all cases of weighted trapdoors. To the best of our knowledge, no such schemes over encrypted data have been proposed to solve this kind of problem. So, determining how to complete a secure weighted similarity search is challengeable.

In this paper, for the first time, we described the problem associated with secure weighted similarity search in the ciphertext domain, and we proposed a new, secure index based on LSH to address this problem. The LSH values are treated as 'keywords' in our SE scheme. The weight of each attribute in a searched data item would be encrypted since it is sensitive information. In addition, in order to rank the final results as well as protecting the privacy of the relevance between the weighted queries and the original data, we propose an algorithm to securely compute the relevant scores, which allows the cloud server to return the most relevant results without revealing any sensitive information. Extensive experiments on real datasets have shown that our scheme is effective and efficient.

The main contributions of this paper are summarised as follows:

We described the problem associated with the secure weighted similarity search in the ciphertext domain and proposed a solution for the problem.

We proposed a secure ranked algorithm so that the cloud server can return the most relevant results without revealing any sensitive information.

We analysed our scheme and proved that it is secure, and we performed extensive experiments on actual datasets to show that our scheme is effective and efficient.

The remaining of this paper is structured as follows. Section 2 introduces the related work, and the problem statement is presented in Section 3. And the entire scheme is demonstrated using an example in Section 4. The security of our scheme is analysed in Section 5, the results of our experiments are presented in Section 6, and our conclusions concerning the scheme are presented in Section 7.

2 Related work

Various protocols and security definitions have been proposed for SE. Traditional SE has been studied extensively as a cryptographic primitive. Song *et al.* [1] first proposed the concept of SE, and they encrypted each word of a document. In their scheme, the search time is directly proportional to the size of the data collection. Their construction supports the insertion and deletion functions in a straightforward method. Then, Goh [2] develops a per-file index design via a Bloom filter and proposed a definition of security for searchable symmetric encryption (SSE). Chang and Mitzenmacher [3] introduced a simulation-based security definition, which was slightly stronger than the definition in [2]. However, these models are impractical for actual scenarios because of the heavy computation overhead.

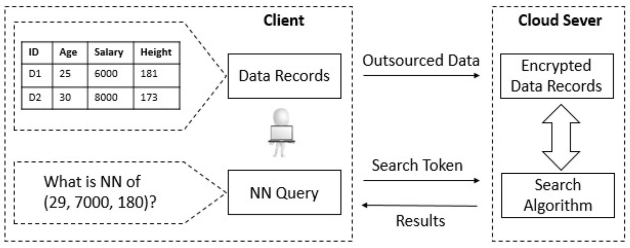


Fig. 1 *Example of a weighted similarity search*

Curtmola *et al.* [4] presented the first index-based SSE solution for single-keyword search with sub-linear search time. They also proposed a definition for adaptive security and proposed new constructions against non-adaptive and adaptive chosen-keyword attacks. Boneh *et al.* [5] proposed the first public key scheme for keyword search in encrypted data. Then, the conjunctive keyword search [6, 7], subset query [8], and range query [8, 9] were studied for the searchable public-key encryption model.

Guo *et al.* [10] applied a single-keyword SSE scheme in the e-healthcare system. Cao *et al.* [11, 12], Sun *et al.* [13, 14], Xia *et al.* [15], and Orencik *et al.* [16] extended the secure single-keyword approach for multi-keyword queries. Zhang *et al.* [17] proposed a scheme for multiple data owners that can support ranked multi-keyword search. In their scheme, the different data owners encrypt their data using their own secret keys. After authentication, the data user can perform a multi-keyword search without knowing any secret keys of the data owners. However, bilinear mapping in this match protocol would cause a large computation overhead. In addition, this scheme does not support dynamic updating of data.

With the development of computer science, multimedia databases, such as images, videos, and web pages, usually are represented as high-dimensional records. Thus, finding similar records or nearest neighbours is much more common and practical, especially in large datasets. Fu *et al.* [18] proposed a similarity search method for encrypted documents based on sim-hash. In order to scale properly for large data sources, they also built a trie-based index to improve the efficiency of searches. Tao *et al.* [19] identified the nearest neighbour search problem in high-dimensional space in the plaintext domain. They took advantage of LSH in their scheme to achieve sub-linear search time. As an improvement, they also constructed an index called LSB-forest, which consisted of a lot of LSB-trees. Wang *et al.* [20] used a wildcard to solve similar keyword search measured by the edit distance. Elmehdwi *et al.* [21] proposed a scheme that can solve the secure kNN search problem. It was the first scheme that could protect the access control in similarity search under high-dimensional data. Their scheme can compare two encrypted vectors without revealing any information. Based on the contribution of [19], lots of tasks have been completed using LSH. Yuan *et al.* [22] proposed a similar search scheme that achieved a constant search time and good accuracy for large-scale data. They regarded the LSH values as keywords and used multiple choice hashing [23, 24] open addressing [23], and cuckoo hashing [25] to get high performance. Cui *et al.* [26] proposed a SE scheme for multiple data owners that can support similarity search. They also used bilinear mapping to let users complete their queries without knowing any information about the data owners' secret keys. Wang *et al.* [27] utilised the R-tree structure to build an efficient index. To compare the location information, they used Order-Preserving Encryption [28, 29] to encrypt the node of R-tree. Instead of generating a temporary circle in the kNN algorithm, they computed a temporary rectangle for easy encryption and comparisons.

3 Problem statement

In this section, first, we propose a new system model to achieve our goals. Then, we discuss the threat model according to the system model. Finally, we illustrate the design goals and the preliminary of our scheme.

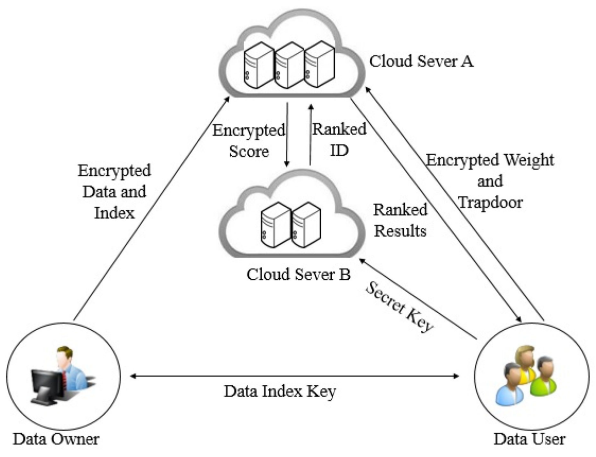


Fig. 2 *System model*

3.1 System model

There are three entities in a traditional SE system model. However, to perform the secure ranked search operation in our scheme, we added another server in our SSE system, as shown in Fig. 2. The system model consists of data owner, data user, cloud server A, and cloud server B. Considering a realistic scenario, the data owner intends to outsource the data to the cloud server so that he can enjoy the high quality service provided by the cloud in hosting his data. Since cloud servers are always employed in an untrusted environment provided by third-party service providers, the data owner must encrypt his or her data to avoid data leakage and unauthorised access. To make the query more convenient and efficient, the data owner will build an encrypted index and outsource it to the cloud server along with the encrypted data. When a data user wants to perform a search operation, first, he or she will generate a trapdoor. Usually, a data user has some requirements about the attributes in the query to get a more accurate result. The data user may consider that one of the attributes is the most important or some attributes are more important than others. So, the user will assign the weights to the trapdoor and send it to the cloud server. The weight information related to the trapdoor should be encrypted to prevent information leakage. As long as the cloud server receives the trapdoor, it will perform the match algorithm and compute the relevant scores, which are encrypted against the relevance information leakage. However, encrypted relevant scores cannot be compared. Hence, we add another cloud server to decrypt and rank the relevant scores. Finally, the cloud server would return the most k relevant data records as a result of the query.

3.2 Threat model

In our scheme, we assume that the authorisation between the data owner and the data users is done appropriately. We also assume that the data owner and the data users who pass the authorisation are trustworthy. The cloud servers in an untrusted cloud environment can be attacked by outside adversaries who can obtain all of the stored data. We assume that the two cloud servers are ‘curious but honest’, which means that they are curious about obtaining the content of the encrypted data, but they follow the designed protocol exactly. Also, we assume that cloud server A will not collude with cloud server B and that cloud server B will not give incorrect answers to cloud server A.

3.3 Design goals

Our scheme will achieve both the performance goals and the safety goals, which ensure secure weighted similarity search under encrypted data in an untrusted cloud environment.

Weighted similarity search: The cloud server should return the most k relevant data records according to a trapdoor. In addition, data users also can add encrypted weight information into the trapdoor, and the cloud server should return more accurate results.

Index privacy: Two aspects of index privacy should be preserved, i.e. (i) the cloud server cannot learn the content of the index because the index directly reflects the content of the data records and (ii) the cloud server cannot deduce any content of the data records by analysing the encrypted index.

Trapdoor privacy: The trapdoor is generated by a searched data item and can require the cloud server to perform a search operation. The trapdoor will preserve the data users’ query information against the cloud server. In addition, the cloud server cannot tell whether a query is a duplicate of an earlier query because it will not receive two identical trapdoors even if two queries are the same.

Relevance score privacy: The relevance score is used to measure the similarity between the search request and data records stored in the cloud. Given the relevance scores, the cloud server that stores the original dataset will learn nothing about the relevance information. The cloud server that owns the secret key can decrypt the scores, but it cannot obtain the original data records.

3.4 Preliminary

Locality-sensitive hashing (LSH) [30]*:* Given a distance r, approximation ratio c, probability values p_1 and p_2 such that $p_1 > p_2$, a hash function $h()$ is (r, cr, p_1, p_2) local sensitive if it satisfies both conditions below:

i. If $\| o_1, o_2 \| < r$, then Pr $[h(o_1) = h(o_2)] \geq p_1$;
ii. If $\| o_1, o_2 \| > cr$, then Pr $[h(o_1) = h(o_2)] \leq p_2$.

LSH functions are known for many distance metrics. For l_p norm, a popular LSH function is defined as follows:

$$h(o) = \left\lfloor \frac{\boldsymbol{a} \cdot \boldsymbol{o} + b}{w} \right\rfloor$$

Here, $\boldsymbol{o}$ represents the d-dimensional vector representation of a point o; $\boldsymbol{a}$ is another d-dimensional vector that each dimension is generated independently by a p-stable distribution; $\boldsymbol{a} \cdot \boldsymbol{o}$ denotes the dot product of two vectors. w is the width of a bucket. b is uniformly drawn from $[o, w)$.

Paillier cryptosystem [31]*:* The Paillier cryptosystem is a homomorphic and probabilistic asymmetric encryption scheme. The secret key sk for decryption is given by two large prime numbers (p, q). The public key pk for encryption is given by (N, g), where N is a product of p and q, and g is in $Z^*_{N^2}$. Let E_{pk} denote the encryption function, and let D_{sk} denote the decryption function. Given $a, b \in Z_N$, the Paillier encryption scheme has the following properties:

Homomorphic addition

$$E_{pk}(a + b) \leftarrow E_{pk}(a) \times E_{pk}(b) \bmod N^2$$

Homomorphic multiplication

$$E_{pk}(a \times b) \leftarrow E_{pk}(a)^b \bmod N^2$$

4 Our proposed scheme

In this section, we introduced the main scheme for secure weighted similarity search under encrypted data records. Our core design is to return a more accurate result based on the data user's requirement. Below, some notations are provided to make you clear in our scheme. Then, we present our design rationale before providing the details.

4.1 Notations

This section gives the notations used throughout the paper. The notation, t, represents an m-dimensional data record, and $[t]$ is the ciphertext of t. T is a record set, i.e. $\{t_1, t_2, \ldots, t_n\}$, and n is the cardinality of the set. We denoted

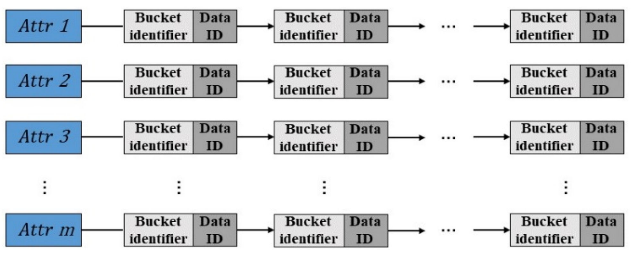


Fig. 3 *Structure of the index*

```
1: $K \leftarrow GenKey(1^\lambda)$
2: for $1 \le j \le m$ do:
3:     for $1 \le i \le n$ do:
4:         Bucket identifier $\leftarrow$ LSH$(t_{i,j})$
5:         If Bucket identifier $\in$ $Attr\ j$:
6:             $Attr\ j$[Bucket identifier]$.add(i)$
7:         else:
8:             $i \leftarrow Attr\ j$[Bucket identifier]
9:     end for
10: end for
11: [Bucket identifier] $\leftarrow Enc$(Bucket identifier, $K$)
    for all Bucket identifier $\in$ Index
```

Fig. 4 *Algorithm 1: Build (K, T)*

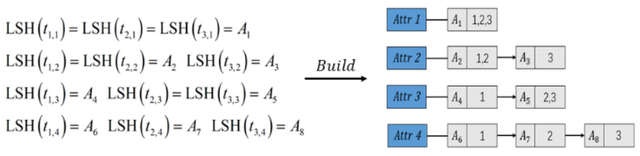

$\mathrm{LSH}(t_{1,1}) = \mathrm{LSH}(t_{2,1}) = \mathrm{LSH}(t_{3,1}) = A_1$
$\mathrm{LSH}(t_{1,2}) = \mathrm{LSH}(t_{2,2}) = A_2$ $\mathrm{LSH}(t_{3,2}) = A_3$
$\mathrm{LSH}(t_{1,3}) = A_4$ $\mathrm{LSH}(t_{2,3}) = \mathrm{LSH}(t_{3,3}) = A_5$
$\mathrm{LSH}(t_{1,4}) = A_6$ $\mathrm{LSH}(t_{2,4}) = A_7$ $\mathrm{LSH}(t_{3,4}) = A_8$
Build
Attr 1
A_1 1,2,3
Attr 2
A_2 1,2
A_3 3
Attr 3
A_4 1
A_5 2,3
Attr 4
A_6 1
A_7 2
A_8 3

Fig. 5 *Example of index building*

$I = \{Attr\,1, Attr\,2, \ldots, Attr\,j, \ldots, Attr\,m\}$ as the encrypted index, where $Attr\ j$ is the jth hash table in the index, and m is the number of attributes in each data record. A is a bucket identifier that is equal to the encrypted LSH value, and $\boldsymbol{V}$ is a set of m-dimensional vectors denoted as $\{v_1, v_2, \ldots, v_n\}$. We defined $q = \{q_1, q_2, \ldots, q_m\}$ as the query point, and we defined $W = \{W_1, W_2, \ldots, W_m\}$ as the weights of q. *Tpdr* is the trapdoor in our scheme, which is generated by q and W. *Score* is the relevance score between q and t. Our scheme contains the functions for the generation of the key, construction of the index, and the processing of the query. The definitions with specified inputs and outputs follow:

$K \leftarrow GenKey(1^\lambda)$ takes a security parameter, λ, as input and outputs a secret key, K.
$I \leftarrow Build(K, T)$ takes K and a record set T as inputs and outputs an encrypted index, I.
$Tpdr \leftarrow GenTpdr(K, q)$ takes K and q as inputs, and outputs a trapdoor, $Tpdr$.
$t \leftarrow Search(I, Tpdr)$ takes I and $Tpdr$ as inputs, and outputs a set of data records, t.

4.2 Design rationale

Inspired by the previous work, we treated the LSH values as keywords and used any known SSE scheme to ensure that the secure similarity search was correct. The data records were mapped into a hash table by an LSH function. Any two data records would be regarded as similar if their LSH values were the same. However, this operation would result in losing the similarity information of each attribute. When the data user has a requirement that some attributes of his or her query are more important than others, it would be difficult for the cloud server to return a more accurate result to meet the requirement. In addition, there are innumerable permutations of weighted queries, while the indices based on LSH are always static. So, it is almost impossible to complete a weighted similarity search under encrypted data by using traditional LSH indices.

To address the above issues, we proposed a newly advanced index, as shown in Fig. 3, which aims to inherit the design benefits of LSH indices as well as conducting a more accurate query. In particular, our index consists of m LSH tables, where m is the number of attributes in each data record. There are many key-values pairs in one hash table. The keys of $Attr\ j$ are bucket identifiers of the jth attribute for all data records, and the values are the IDs of the corresponding data records. Instead of focusing on the similarity between two data records, we computed the LSH value for each attribute. Therefore, we evaluated the similarity of data records by each attribute. To evaluate the relevance of the data records, we proposed n m-dimensional vectors, called relevance vectors, where n is the number of data records. To compute the encrypted relevance scores, we employed Paillier cryptosystem and identify another cloud server to decrypt and compare the scores.

4.3 Details of the main scheme

Build: The data owner owns the original database (T) of n records. A data record consists of m attributes. So, $t_{i,j}$ means the jth attribute of the data record, t_i. To enable efficient weighted similarity search operations on these data records, which have been encrypted, the data owner would build a secure searchable index, I.

As shown in Algorithm 1 (see Fig. 4), first, we generated a secret key, K, to encrypt the index. The problem we had to solve is how to perform a secure weighted similarity search. If we built a traditional index based on LSH, which is used to evaluate the similarity between two data records, the information about each attribute would be mixed. We cannot extract any similar information between attributes. Furthermore, because LSH functions are not cryptographic hash functions, the LSH values should be encrypted to protect the privacy of the original dataset. It is far more difficult to find any similar information about attributes through the encrypted values. So, a traditional index based on LSH cannot be used to complete the weighted similarity search operations.

To address this problem, we computed the LSH value called the 'bucket identifier' of each attribute instead of computing the LSH value of the entire data record. Then, the two data records are treated as similar in one attribute if their LSH values are the same. The index will be divided into m hash tables based on the number of attributes, where m is the number of attributes in one data record. The keys in the hash table are the bucket identifiers, and the values are the corresponding IDs of the data records.

Then, we inserted the key-value pairs into the index. If a key for one attribute has already been in the hash table, the value would be inserted into the corresponding bucket. However, if the key is not in the hash table, the key-value pair would be inserted into the end of the hash table. Fig. 5 shows a small example of building our index. The index is built by three data records, i.e. t_1, t_2, t_3, and each data record has four attributes. Fig. 5 also shows the relationships between the LSH values. Finally, we encrypted all of the bucket identifiers to protect the security of the index by using the secret key, K.

Generate trapdoor: Once a data user wants to perform a search operation in the encrypted data that are stored in the cloud server, he/she should generate a trapdoor based on the query and the weights of each attribute in the query.

The main steps are given in Algorithm 2 (see Fig. 6). First, the data user computes the LSH values of the query attribute-wise. To execute the match algorithm successfully as well as protecting the privacy of the query, the LSH values will be encrypted by the index generation key, i.e. $Enc(\mathrm{LSH}(q)) = \langle Enc(\mathrm{LSH}(q_1)), Enc(\mathrm{LSH}(q_2)), \ldots, Enc(\mathrm{LSH}(q_m))\rangle$. Enc will be computed and sent to cloud A. To rank the similarity between the query and the data records while preserving the privacy of the relevance information, the data user also encrypts the weights of the attributes in the

```
1: for $1 \le j \le m$ do:
2:     $[\mathrm{LSH}(q_j)] \leftarrow Enc(\mathrm{LSH}(q_j))$
3:     compute $E_{pk}(W_j)$
4: end for
5: $Tpdr_1 \leftarrow [LSH(q)]$
6: $Tpdr_2 \leftarrow E_{pk}(W)$
7: send $Tpdr = \{Tpdr_1, Tpdr_2\}$ to the cloud server A
8: send $pk$ to the cloud server B
```

Fig. 6 *Algorithm 2: GenTpdr (K, q)*

```
Cloud server A:
 1: for $1 \le j \le m$ do:
 2:     If $Tpdr_{1,j}$ in $Attr\ j$
 3:     updata $V_{ID,j}$ from 0 to 1 where ID is in $Attr\ j$
 4: end for
 5: for $1 \le i \le n$ do:
 6:     compute $E_{pk}(Score_i)$
 7: end for
Cloud server B:
 8: Reveive $E_{pk}(Score)$ from cloud server A
 9: for $1 \le i \le n$ do:
10:     $D_{sk}(E_{pk}(Score_i))$
11: end for
12: Generate an ID list $\delta$
Cloud server A:
13: Receive ID list $\delta$ from cloud server A
14: Return the corresponding data records
```

Fig. 7 *Algorithm 3: Search (I, Tpdr)*

query. However, the data encrypted by symmetric encryption cannot be computed. To solve this problem, we used another encryption cryptosystem, i.e. the Paillier cryptosystem, which has an advantage that the encrypted data can be computed. To be more specific, the data encrypted by the Paillier cryptosystem can be added and multiplied without decryption. The data user computes $E_{pk}(W) = \langle E_{pk}(W_1), E_{pk}(W_2), \dots, E_{pk}(W_m)\rangle$ and sends it to the cloud server A for further computation. Cloud server A can compute the encrypted relevance scores and return the encrypted results. Anyone who owns the secret key can decrypt them and obtain the plaintext results.

In each query, the user should generate a pair of keys, i.e. a secret key, *sk*, and a public key, *pk*. The public key is used to encrypt the weights of his query and the secret key is used to decrypt the results. But an additional problem is that the encrypted results cannot be compared. A straightforward solution is that the cloud server can return all of the encrypted results to the data user. The data user decrypts the results and compares them to obtain a ranked sequence. However, this solution will cause a lot of communication overhead and computing overhead. Also, the SE scheme should keep the user's experience essentially the same as it was in the plaintext domain, i.e. when a user sends a trapdoor, the cloud server should autonomously respond with the corresponding results, ideally without multiple rounds of user interactions or extra computing operations. To ensure that this happens, we identified another cloud server, cloud server B, and the former cloud server is called cloud server A. This approach allows the user to share the secret key with cloud server B without worrying about the privacy leakage.

When the trapdoor is submitted to cloud server A by the user, the user's secret key would be sent to cloud server B. According to the match algorithm, cloud server A first matches $Tpdr_1$ with the index. Then, cloud server A computes the encrypted relevance scores and sends them to cloud server B. With the secret key, cloud server B can decrypt and compare the scores. Finally, the ranked sequence will be outsourced to the cloud server A. Although the data user shares the secret key with cloud server B, the only information that is made available to cloud server B is the encrypted results and the corresponding ID of the data records. These interactions between cloud server A and cloud server B will not reveal any security information related to the origin dataset.

Search: The trapdoor consists of the query part, denoted as $Tpdr_1$, and the weight part, denoted as $Tpdr_2$. Upon receiving the trapdoor, with the private input $\langle Tpdr_1, Tpdr_2\rangle$, the output of this step, denoted by $E_{pk}(Score)$, is the encryption of the relevance scores which are used to measure the similarity between the query and the data records. Essentially, $E_{pk}(Score)$ are the encryption of the inner product between $Tpdr_2$ and the data records.

In the details of Algorithm 3 (see Fig. 7), cloud server A traverses the index to search whether $Tpdr_{1,j}$ is in it, where $Tpdr_{1,j}$ means the *j*th attribute in the query part of the trapdoor. It is a deterministic encryption primitive to encrypt the index and $Tpdr_1$. Therefore, the equality between the encrypted bucket identifier and $Tpdr_1$ can be regarded that they are equal in the plaintext domain. If $Tpdr_{1,j}$ matches a bucket identifier in *Attr j*, cloud server A will obtain the corresponding IDs of the data records located in the bucket. The match indicates that the query in the trapdoor is similar to the data records on the *j*th attribute.

To conduct a secure weighted similarity search, we needed an extra implement known as the master relevance vector. The master relevance vector consists of *n* *m*-dimensional sub relevance vectors, where *n* is the number of data records, and *m* is the number of attributes. This indicates that each relevance vector is connected with one data record. We initialled the relevance vector all ‘0’ for each component. After the match algorithm, some ‘0s’ would be updated by ‘1.’ We regarded ‘0’ as dissimilar and ‘1’ as similar with the corresponding attribute. As mentioned above, the cloud server A would obtain some IDs of data records after the $Tpdr_{1,j}$ matching. Then, cloud server A updates ‘0’ to ‘1’ in relevance vectors according to the IDs on the *j*th component. Fig. 8 shows an example of this. After updating, cloud server A securely computes the relevance scores as follows: (see equation below) where $Score_i$ is the secure relevance score between the weighted query and data record t_i, $Tpdr_{2,j}$ is the *j*th dimension of the weight part in the trapdoor, and $v_{i,j}$ is the *j*th component of relevance vector, $\boldsymbol{v}_i$. The result of this algorithm is an encrypted inner product.

Due to the fact that encrypted values cannot be compared, cloud server A cannot evaluate the similarity even if it obtains the

$$\begin{aligned}
&E_{pk}(Score_i)\\
&= \left(E_{pk}(Tpdr_{2,1})^{v_{i,1}} \times E_{pk}(Tpdr_{2,2})^{v_{i,2}} \times \cdots \times E_{pk}(Tpdr_{2,m})^{v_{i,m}}\right)\\
&= E_{pk}(Tpdr_{2,1}\cdot v_{i,1}) \times E_{pk}(Tpdr_{2,2}\cdot v_{i,2}) \times \cdots \times E_{pk}(Tpdr_{2,m}\cdot v_{i,m})\\
&= E_{pk}\left(\sum_{j=1}^{m} Tpdr_{2,j} \times v_{i,j}\right)\\
&= E_{pk}(Tpdr_2 \cdot \boldsymbol{v}_i)
\end{aligned}$$

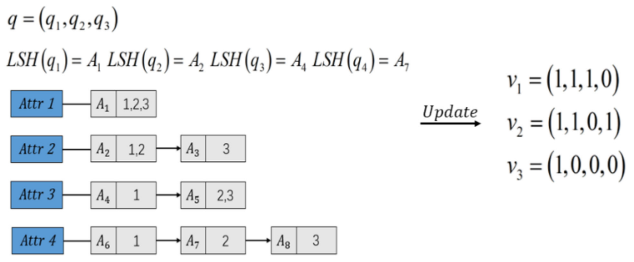


Fig. 8 *Example of updating the relevance vector*

encrypted inner product between the weighted query and data records. Cloud server A will send $\{\langle 1, E_{pk}(Score_1)\rangle, \langle 2, E_{pk}(Score_2)\rangle, \ldots, \langle n, E_{pk}(Score_n)\rangle\}$ to cloud server B, where the entry $\langle i, E_{pk}(Score_i)\rangle$ corresponds to the data record t_i. When the results are received, cloud server B decrypts the encrypted relevance scores in each entry to get $Score_i = D_{sk}(E_{pk}(Score_i))$. Then, cloud server B generates a list, $\delta = \langle i_1, i_2, \ldots, i_k\rangle$, such that $\langle Score_{i_1}, Score_{i_2}, \ldots, Score_{i_k}\rangle$ are the top k largest relevance scores among $\langle d_1, d_2 \ldots, d_n\rangle$. After this, cloud server B sends δ to cloud server A. Finally, cloud server A returns the user encrypted data records that correspond to δ.

5 Security analysis

Our scheme can be used to conduct a weighted similarity search by taking advantage of the techniques from SSE. We treat LSH values as keywords, and as a result, our scheme has the same limitation as for the previous SSE schemes. In this section, we provide a step-by-step description of the security analyses to demonstrate that the security requirements have been satisfied.

5.1 Security of our scheme

The definition of security that has been used for SSE, which was introduced by Goh [2], is indistinguishability against chosen-keyword attacks (IND2-CKA).

Intuitively, the security guarantees that IND2-CKA provides can be described as follows, i.e. given access to a set of indices, the adversary cannot learn any partial information about the underlying dataset.

Game 1 (IND2-CKA): Let $SI = (GenKey, Build, GenTpdr, Search)$ be a secure index scheme, Δ be a dictionary, A be an adversary, and consider the following experiment $CKA_{SI,A}(\lambda)$:

i. A generates a collection of n datasets $T = (T_1, T_2, \ldots, T_n)$ from Δ.
ii. The challenger generates a key $K \leftarrow GenKey(\lambda)$ and indices $(I_1, I_2, \ldots, I_k)$, such that $I_i \leftarrow Index(T_i)$.
iii. Given $(I_1, I_2, \ldots, I_k)$ and oracle access to $GenTpdr()$, A outputs two datasets, T_0^* and T_1^*, such that $T_0^* \in T$, $T_1^* \subseteq \Delta$, $|T_0^* \setminus T_1^*| \neq 0$ and $|T_1^* \setminus T_0^*| \neq 0$.
iv. The challenger chooses a bit, $b \in \{0, 1\}$, uniformly at random and computes $I_b = Build(T_b)$.
v. Given I_b and oracle access to $GenTpdr()$, A outputs a bit b'.
vi. The output of the experiment is 1 if $b' = b$; otherwise it is 0.
vii. We say that SI is IND2-CKA secure if, for all polynomial-sized adversaries A, $\Pr[CKA_{SI,A}(\lambda) = 1] \leq (1/2) + negl(\lambda)$, where the probability is over the choice of b and the coins of $GenKey()$ and $Enc()$.

5.2 Security of the relevance score

Before analysing the security of the relevance score, first, we introduce homomorphic encryption (HE). The most common definition is as follows: Let M denote the set of plaintext and let C denote the set of ciphertext. An encryption scheme is said to be homomorphic if, for any given encryption key, k, the encryption function, E, satisfies

$$\forall m_1, m_2 \in M, \quad E(m_1 \odot_M m_2) \leftarrow E(m_1) \odot_C E(m_2)$$

for some operations $\odot_M$ in M and $\odot_C$ in C.

It should be emphasised that, when we know c, we can compute $c' = c \odot_C c$ and deduce that c' is the ciphertext of $m' = m \odot_M m$. According to the previous remark on adaptive chosen-ciphertext indistinguishability, a HE has no access to the strongest security requirement. The highest security level it can reach is IND-CPA. And it has been proven that the Paillier cryptosystem achieves the highest security level for the HE scheme. The relevance scores of our scheme are secure since they are encrypted and computed by the Paillier cryptosystem.

5.3 Search pattern and access pattern

Specifically, cloud server A initially obtains some default attributes of the index without performing any search operation, i.e. the attributes of the data records, the capacity of the encrypted index, and which ID is similar to others. When the search begins, the access pattern and the search pattern are subsequently revealed. To be more specific, the access pattern in our scheme includes a set of IDs of similar data records and the accessed buckets of the index. For the search pattern, a notion from SSE for a keyword search is whether the same encrypted data have been retrieved by two different trapdoors.

In the keyword search, each query generates a single trapdoor, and the search pattern is revealed by those deterministic trapdoors. A repeated trapdoor means that the query has been searched before. For our scheme, less information will be revealed. Each search request generates a trapdoor that consists of two sub trapdoors. The first part of the trapdoor, i.e. the information about the query, is encrypted by AES. For the second part, the weight information is encrypted by the Paillier cryptosystem. This means that two identical query requests will not generate identical trapdoors. Therefore, the search pattern will not be revealed.

The problem our scheme solves is that it allows a weighted similarity search to be conducted. For a fixed result, there may be several search requests since it is not an exact search. So, the cloud server cannot obtain any useful information about the index and trapdoor from the results of a query. Therefore, the query pattern will not be revealed either.

6 Performance evaluation

In this section, we present the experimental evaluation of the proposed scheme. We conducted the experiment with both actual datasets, i.e. the UCI Machine Learning Repository [32] and a synthetic dataset. The experiment programs were coded by the Python programming language on a PC. For cryptographic primitives, we implemented symmetric encryption via AES-128, and the key size of the Paillier encryption was 128. In the original dataset, all the values of attributes are positive numbers. For the weights of the attributes, the user can assign any positive integer to them. Like most SSE schemes, we evaluated the performance from two aspects, i.e. the cost for index construction and the cost of search. And we compared our scheme with two other schemes, a basic scheme and a full scheme, proposed in Secure kNN Query over Encrypted Data in Outsourced Environments [21] by the search time.

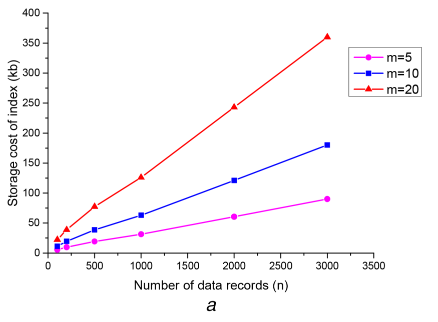
6.1 Construction of the index

In our scheme, we proposed a new secure advanced index based on LSH. In this subsection, we analyse the cost of storing the indices by comparing the numbers of data records, i.e. varying the number of data records (n) and the number of attributes (m).

Table 1 shows that, given the same dimensions of data records ($m = 10$), at first, the storage cost for an encrypted index doubles the storage cost for a dataset in the plaintext domain. The encrypted index consists of the encrypted LSH values only, which are called bucket identifiers, and the corresponding IDs of the data records stored in the buckets. Keeping the number of dimensions constant, the increasing rate of storage cost for the encrypted index will become significantly lower than the rate for a dataset in the plaintext domain as the number of data records increases. Eventually, the storage cost of the index should be smaller than that of the dataset due to the utilising of LSH. Fig. 9*a* shows the storage cost of the encrypted index for different dimensions. The figure demonstrates that the storage cost for our scheme increased linearly with the increasing number of data records. Fig. 9*b* shows that, for different number of data records, the storage cost for our scheme also increased linearly as the number of dimensions increased. Fig. 10 shows the time cost of our advanced index.

Table 1 Storage requirement (kb) for $m = 10$

Number, n	Datasets	Indices
100	3.66	11.1
200	7.36	19.4
500	18.1	38.6
1000	36.3	63
2000	72.6	121
3000	109	180



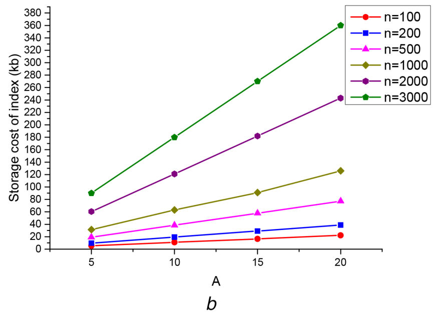


Fig. 9 *Storage cost of indices*

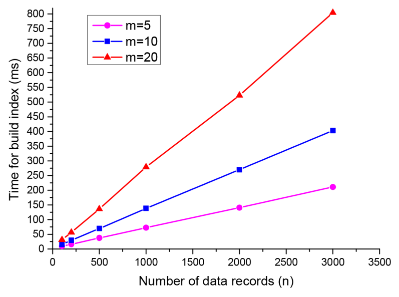


Fig. 10 *Time cost of indices*

The data owner should traverse all of the data records when building an index. Therefore, the time cost also will increase linearly with an increasing number of data records.

6.2 Search time for our scheme

The search operation has two steps. Step 1 is that the cloud server searches the encrypted index according to the query part of the trapdoor. Fig. 11*a* shows that the time required for query step 1 increased sub-linearly as the number of data records increased. The fundamental reason is that it is an LSH-based index in our scheme. By utilising the LSH-based index, the cloud server does not traverse all of the data records when performing a search operation. It is identical to searching in a hash table. Step 2 is that the cloud server computes the secure inner product using HE. Fig. 11*b* shows that, for different dimensions, the time required for performing step 2 increases linearly as the number of data records increases. However, there are some disadvantages associated with using HE, one of which is the efficiency problem. It is a very time-consuming cryptosystem because the computing occurs in the ciphertext domain. As a result, the time for step 2 is much longer that the time for step 1. So, the entire time required to conduct a search operation essentially has a linear relationship with the number of data records.

Fig. 11*c* further demonstrates this conclusion by keeping the number of dimensions constant. Also, it shows that the time required to search can be influenced by the key size of the Paillier encryption. As shown in Fig. 11*d*, by fixing $m = 5$ and $n = 500$, we evaluated the time cost of a query for various key sizes of the Paillier encryption and various number of nearest neighbours. It is demonstrated that the number of nearest neighbours had no effect on the time required to search. The time cost was almost constant if the number of m, n, and K were constant. In addition, as the key size of the Paillier encryption increases, our scheme would be more and more secure. But it acts as a trade-off between security and efficiency. So, a key size should be determined for the Paillier encryption that provides a good balance between security and efficiency.

There are two rounds for the interaction between cloud servers A and B. Round 1 is when a cloud server A sends $\{\langle 1, E_{pk}(Score_1)\rangle, \langle 2, E_{pk}(Score_2)\rangle, \dots, \langle n, E_{pk}(Score_n)\rangle\}$ to cloud server B. By keeping the size of the key for Paillier encryption constant, Fig. 12*a* shows that the communication cost for round 1 increases linearly with an increasing number of data records. It also shows that, for the same number of data records, the communication cost has nothing to do with the number of attributes, since the bits of $E_{pk}(Score_i)$ are influenced only by the size of the key. Fig. 12*b* demonstrates this conclusion further by investigating the effect of different sizes of keys for the Paillier encryption. Round 2 represents that cloud server B sends the ID list $\delta = \langle i_1, i_2, \dots, i_k\rangle$ to cloud server A. We do not consider the community cost of this step since it is only related to the number of nearest neighbours, and it only causes a little overhead.

6.3 Construction of the index

In this section, we evaluated the efficiency by the search time of our proposed scheme and two schemes proposed in [21]. All of the three schemes employed the same system model. We analysed the search time by varying the number of data records, the number of attributes and the number of nearest neighbours.

As shown in Fig. 13*a* by fixing $m = 5$ and $k = 5$, in Fig. 13*b* by fixing $n = 100$ and $k = 5$, in Fig. 13*c* by fixing $n = 100$ and $m = 5$, we compared the search time by varying different parameters. They are demonstrated that our proposed scheme is extremely more efficient than the other schemes due to the fact that we employed the LSH instead of the Euclidean distance to evaluate the similarity between the trapdoor and the original data records. The two protocols act as a trade-off between accuracy and efficiency. As a result, our scheme is less accurate in an ordinary similarity search. However, the most significant contribution in this paper is that the data user can assign weights for his/her trapdoor. Our scheme can achieve greater accuracy in a secure weighted similarity search.

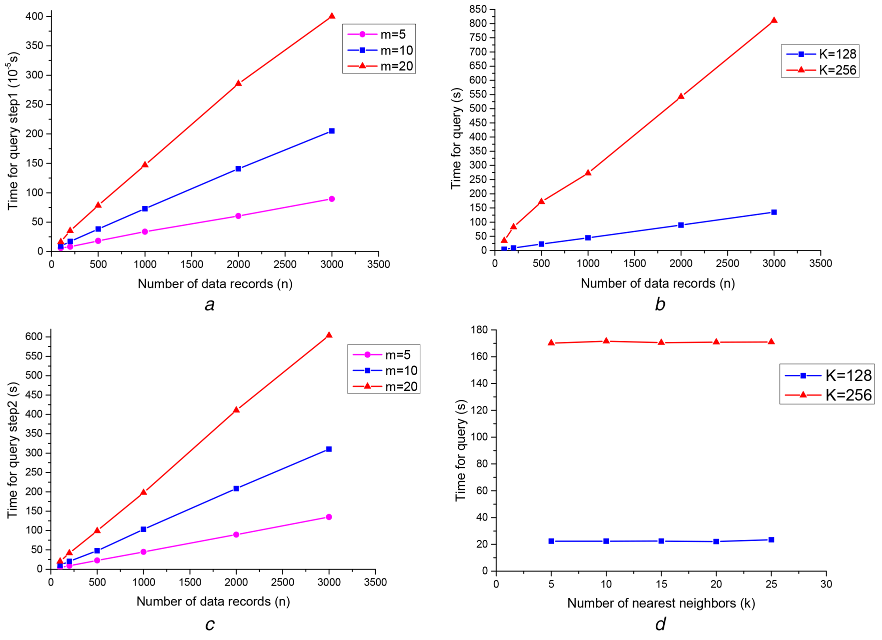


Fig. 11 *Time required for searching*

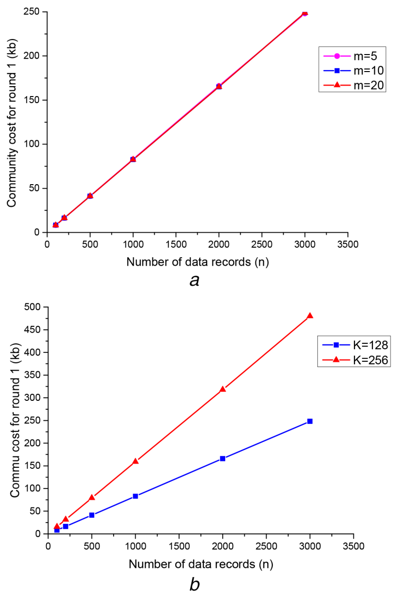


Fig. 12 *Communication cost of our scheme*

7 Conclusions

In this paper, we explored the problem of SE in the untrusted cloud computing environment. Different from prior works, our scheme enables a secure and convenient weighted similarity search. Our design starts with two building blocks, LSH and SSE. Since our target was a reasonable requirement, we transformed a traditional LSH-based index. To achieve secure relevance scores computing and comparing, we identified another cloud server and used the Paillier cryptosystem. By adapting the security framework of SSE, we carefully identified any information leakage and proved the security of our scheme. In our future work, we plan to improve our scheme to support similarity search for multiple data owners.

8 Acknowledgments

This paper was supported by the National Natural Science Foundation of China under grant nos. 61501080 and 61572095, and the Fundamental Research Funds for the Central Universities' under no. DUT16QY09.

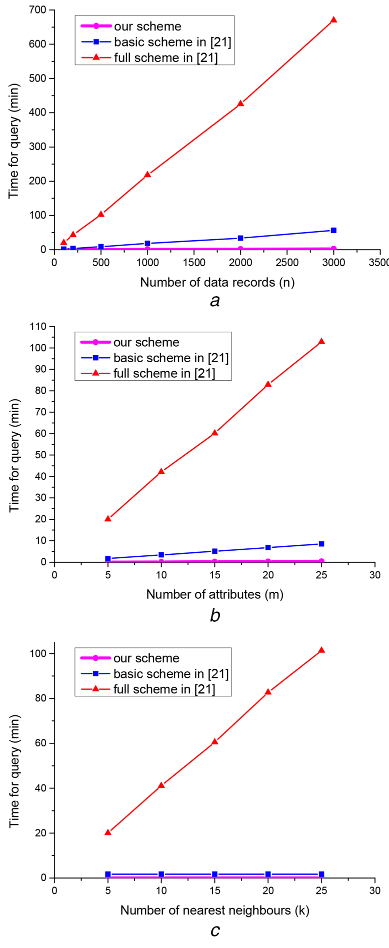


Fig. 13 *Search time for comparing*

9 References

[1] Song, D.X., Wagner, D., Perrig, A.: 'Practical techniques for searches on encrypted data'. Security and Privacy, Oakland, CA, USA, 2000, pp. 44–55

[2] Goh, E.-J.: 'Secure indexes', IACR Cryptology ePrint Archive, 2003, p. 216

[3] Chang, Y.C., Mitzenmacher, M.: 'Privacy preserving keyword searches on remote encrypted data'. Proc. Int. Conf. Applied Cryptography and Network Security, New York City, NY, USA, June 2005, pp. 442–455

[4] Curtmola, R., Garay, J., Kamara, S., *et al.*: 'Searchable symmetric encryption: improved definitions and efficient constructions', *J. Comput. Secur.*, 2011, **19**, (5), pp. 895–934

[5] Boneh, D., Di Crescenzo, G., Ostrovsky, R., *et al.*: 'Public key encryption with keyword search'. Proc. Int. Conf. Eurocrypt, Interlaken, Switzerland, May 2004, pp. 506–522

[6] Golle, P., Staddon, J., Waters, B.: 'Secure conjunctive keyword search over encrypted data'. Proc. Int. Conf. Applied Cryptography and Network Security, Yellow Mountain, China, June 2004, pp. 31–45

[7] Ballard, L., Kamara, S., Monrose, F.: 'Achieving efficient conjunctive keyword searches over encrypted data'. Proc. Int. Conf. Int. Conf. on Information, Communications and Signal Processing, Beijing, China, 2005, pp. 414–426

[8] Boneh, D., Waters, B.: 'Conjunctive, subset, and range queries on encrypted data'. Proc. Int. Conf. Theory of Cryptography, Amsterdam, Netherlands, 2007, pp. 535–554

[9] Shi, E., Bethencourt, J., Chan, T.H., *et al.*: 'Multi-dimensional range query over encrypted data'. Proc. Int. Conf. Security and Privacy, Oakland, CA, USA, May 2007, pp. 350–364

[10] Guo, C., Zhuang, R., Jie, Y., *et al.*: 'Fine-grained database field search using attribute-based encryption for E-healthcare clouds', *J. Med. Syst.*, 2016, **40**, (11), p. 235

[11] Cao, N., Wang, C., Li, M., *et al.*: 'Privacy preserving multi-keyword ranked search over encrypted cloud data'. Proc. Int. Conf. IEEE INFOCOM, Shanghai, China, 2011, pp. 829–837

[12] Cao, N., Wang, C., Li, M., *et al.*: 'Privacy preserving multi-keyword ranked search over encrypted cloud data', *IEEE Trans. Parallel Distrib. Syst.*, 2014, **25**, (1), pp. 222–233

[13] Sun, W., Wang, B., Cao, N., *et al.*: 'Privacy-preserving multi-keyword text search in the cloud supporting similarity-based ranking'. Proc. Int. Conf. IEEE ASIACCS, Hangzhou, China, May 2013, pp. 71–81

[14] Sun, W., Wang, B., Cao, N., *et al.*: 'Verifiable privacy-preserving multi-keyword text search in the cloud supporting similarity-based ranking', *IEEE Trans. Parallel Distrib. Syst.*, 2014, **25**, (11), pp. 3025–3035

[15] Xia, Z., Wang, X., Sun, X., *et al.*: 'A secure and dynamic multi-keyword ranked search scheme over encrypted cloud data', *IEEE Trans. Parallel Distrib. Syst.*, 2016, **27**, (2), pp. 340–352

[16] Orencik, C., Kantarcioglu, M., Savas, E.: 'A practical and secure multi-keyword search method over encrypted cloud data'. Proc. Int. Conf. Cloud Computing (CLOUD), Santa Clara, CA, USA, June 2013, pp. 390–397

[17] Zhang, W., Lin, Y., Xiao, S., *et al.*: 'Privacy preserving ranked multi-keyword search for multiple data owners in cloud computing', *IEEE Trans. Comput.*, 2015, **65**, (5), pp. 1566–1577

[18] Fu, Z.J., Shu, J.G., Wang, J., *et al.*: 'Privacy-preserving smart similarity search based on simhash over encrypted data in cloud computing', *J. Internet Technol.*, 2015, **16**, (3), pp. 453–460

[19] Tao, Y., Yi, K., Sheng, C., *et al.*: 'Quality and efficiency in high dimensional nearest neighbour search'. Proc. Int. Conf. ACM SIGMOD, Providence, RI, USA, June 2009, pp. 563–576

[20] Wang, C., Ren, K., Yu, S., *et al.*: 'Achieving usable and privacy-assured similarity search over outsourced cloud data'. Proc. Int. Conf. IEEE INFOCOM, Orlando, FL, USA, March 2012, pp. 451–459

[21] Elmehdwi, Y., Samanthula, B.K., Jiang, W.: 'Secure k-nearest neighbor query over encrypted data in outsourced environments'. Proc. Int. Conf. IEEE ICDE, Chicago, IL, USA, 2014, pp. 664–675

[22] Yuan, X., Cui, H., Wang, X., *et al.*: 'Enabling privacy-assured similarity retrieval over millions of encrypted records'. Proc. Int. Conf. European Symp. on Research in Computer Security, Vienna, Austria, September 2015, pp. 40–60

[23] Lv, Q., Josephson, W., Wang, Z., *et al.*: 'Multi-probe LSH: efficient indexing for high-dimensional similarity search'. Proc. Int. Conf. VLDB, Vienna Austria, 2007, pp. 950–961

[24] Panigrahy, R.: 'Entropy based nearest neighbour search in high dimensions'. Proc. Int. Conf. ACM-SIAM Symp. on Discrete Algorithm, Miami, FL, USA, January 2006, pp. 1186–1195

[25] Hua, Y., Xiao, B., Liu, X.: 'Nest: locality-aware approximate query service for cloud computing'. Proc. Int. Conf. IEEE INFOCOM, Turin, Italy, April 2013, pp. 1303–1311

[26] Cui, H., Yuan, X., Zheng, Y., *et al.*: 'Enabling secure and effective near-duplicate detection over encrypted in-network storage'. Proc. Int. Conf. IEEE INFOCOM, San Francisco, CA, USA, April 2016, pp. 1–9

[27] Wang, B., Hou, Y., Li, M.: 'Practical and secure nearest neighbour search on encrypted large-scale data'. Proc. Int. Conf. IEEE INFOCOM, San Francisco, CA, USA, April 2016, pp. 1–9

[28] Boldyreva, A., Chenette, N., Lee, Y., *et al.*: 'Order-preserving symmetric encryption'. Proc. Int. Conf. EUROCRYPT, Cologne, Germany, April 2009, pp. 224–241

[29] Popa, R.A., Li, F.H., Zeldovich, N.: 'An ideal-security protocol for order-preserving encoding'. Proc. Int. Conf. IEEE Symp. on Security and Privacy, Berkeley, CA, USA, May 2013, pp. 463–477

[30] Datar, M., Immorlica, N., Indyk, P., *et al.*: 'Locality-sensitive hashing scheme based on p-stable distributions'. Proc. Int. Conf. ACM Symp. on Computational Geometry, Brooklyn, NY, USA, June 2004, pp. 253–262

[31] Paillier, P.: 'Public-key cryptosystems based on composite degree residuosity classes'. Proc. Int. Conf. EUROCRYPT, Prague, Czech Republic, May 1999, pp. 223–238

[32] 'UCI Machine Learning Repository'. Available at https://archive.ics.uci.edu/ml/datasets/ILPD+%28Indian+Liver+Patient+Dataset%29