# Secure Similarity Search over Encrypted Non-Uniform Datasets

Cheng Guo, Member, IEEE, Wanping Liu, Ximeng Liu, Member, IEEE, and Yinghui Zhang, Member, IEEE

**Abstract**—Searchable symmetric encryption (SSE) enables a user to outsource a private dataset to a cloud server in encrypted form while retaining the ability to search over the encrypted outsourced data. The existing SSE schemes improve the search and safety performances from different perspectives. However, almost none of the existing SSE schemes considers the data distribution issues. We find that when the dataset is not distributed uniformly, the search quality based on the conventional methods decreases. Therefore, the existing SSE schemes cannot guarantee high search quality when faced with non-uniform datasets. In addition, most existing SSE solutions cannot hide the distribution of the query set. In this paper, we design a Secure similarity search over Encrypted Non-uniform and high-dimensional Datasets (SEND) with a novel way to enhance security. The basic idea is to combine SSE with locality-sensitive hashing (LSH). Unlike earlier schemes, SEND uses selective hashing, which has better performance for non-uniform datasets. Also, we present a novel approach to hide the distribution of the query set, which makes SEND more secure. Our experimental results indicate SEND achieves a high search quality of recall and precision, and it is proven secure against adaptively chosen query attacks in the standard model.

**Index Terms**—cloud security, similarity search, searchable encryption, selective hashing.

## 1 Introduction

In recent years, cloud computing has been highly praised by the industry and a series of services have been launched based on cloud computing platforms. Large numbers of datasets have been stored on remote cloud servers. The outsouced storage allows both the data owner and the data user to avoid the cumbersome management of data at the local level, and its immediate application provides more convenient access service. However, in such a service, both the data and query requests are exposed to the cloud server. Consequently, this approach raises serious data security and privacy concerns.

To address above issues, data owners often choose to store the dataset in encrypted form on a cloud server. However, the similarity search methods available on the plaintext dataset are no longer applicable on the encrypted dataset. To tackle this problem, searchable symmetric encryption (SSE) [1] has been proposed as a promising technique that enables users to retrieve encrypted data of interest efficiently from a cloud server, and it has attracted widespread attention in many areas, such as multi-function query [2]–[4], multimedia databases [5], and near-duplication detection [6].

Some SSE schemes search for similar texts based on keywords and some search for similar high-dimensional data objects based on high-dimensional data object. In this paper, we focus on the latter. The problem of retrieving encrypted, high-dimensional data from a cloud server is usually handled by a combination of SSE and Locality Sensitive Hashing (LSH), where the LSH hash values of the data are treated as the keywords of SSE. Actually, several researchers [7]–[9] have provided solutions based on this idea. [7] proposes a secure LSH index to implement a secure SSE scheme. The method [8] based on simhash enables data users to find similar encrypted documents by submitting a query document. [9] crafts a high performance encrypted index based on multiple-choice hashing, open hashing and cuckoo hashing.

However, all of the above works handle the problem without considering the distribution of the data. If we directly apply an algorithm that assumes that the data distribution is uniform to the actual data, we cannot achieve the desired result in most cases. In fact, actual distributions of data are often non-uniform and frequently very skewed in practice, such as road intersections [10], star coordinates [10], movie recommendation system (most people rate very few movies or none at all [11]), credit card fraud detection [12] (about 20% of dataset are fraudulent [13]) and even the IP address data [14]. Moreover, the dimensions of the feature vectors in a real dateset are usually correlated and dependent which also exacerbates the non-uniform distribution of a dataset.

In addition, research has focused on developing higher security for SSE. Therefore, it would be desirable to minimize the amount of information that the scheme leaks. The straightforward combination of SSE and LSH schemes regard LSH hash values as search keywords and use the SSE framework to conduct secure similarity searches over the encrypted dataset. In this context, identical LSH hash values indicate repeated query tokens, and the overlapping tokens indicate the similarity of query points. Such leakage may

*Cheng Guo, Wanping Liu are with the School of Software Technology, Dalian University of Technology and Key Laboratory for Ubiquitous Network and Service Software of Liaoning Province, Development Zone, Dalian 116620, China. (e-mail: guocheng@dlut.edu.cn, 18840829505@163.com).*
*Ximeng Liu is with the College of Mathematics and Computer Science, Fuzhou University, China and Fujian Provincial Key Laboratory of Information Security of Network Systems, Fuzhou, China. (e-mail: snbnix@gmail.com).*
*Yinghui Zhang is with National Engineering Laboratory for Wireless Security, Xi'an University of Posts and Telecommunications, Xi'an 710121, P.R. China and Westone Cryptologic Research Center, Beijing 100070, P.R. China. (e-mail: yhzhaang@163.com).*

lead to statistical attacks [15], especially when the adversary has partial or full knowledge of the queries.

To address the above two problems, in this paper, we propose a novel scheme that can be used to query on encrypted, non-uniform and high-dimensional datasets with better search quality and greater security. On one hand, to improve the search quality in the case of non-uniform data distribution, we consider the distribution of the data when constructing index and use selective hashing to achieve this goal. More specifically, we estimate the density around each data point and then select an appropriate radius as its query range to construct the index. On the other hand, to enhance the security, we propose a two-server setting and use a random index to hide the distribution of the query set. The encrypted index associated with the dataset is stored on server $A$, and the random index is stored on server $B$. In this way, our scheme hides the distribution of the query set and provides higher security, and the contributions are summarized as follows:

- We design a **S**ecure similarity search scheme over **E**ncrypted **N**on-uniform **D**ataset (SEND) and SEND has a higher recall and precision than traditional LSH-based SSE.
- We present a new method to hide the distribution of the query set and prevent the cloud server from inferring sensitive information by using statistical methods based on query distribution.
- We define the search distribution and integrate the search distribution into the leak functions, we present the simulation-based security definition, and prove that the scheme is secure against adaptive chosen query attacks.
- We evaluate our constructions using four datasets. The evaluations show that SEND is efficient with respect to the time required, and the records that are retrieved have good precision and recall.

The rest of the paper is organized as follows. Related work is summarized in Section 2. Section 3 gives some of the preliminary knowledge used in our scheme. Section 4 shows the system model and security definitions. Then, we present the proposed scheme in Section 5, and we formally define the leakage functions and prove that our scheme is secure against adaptive chosen query attacks in Section 6. Section 7 presents our experimental results, and our conclusions are presented in Section 8.

## 2 RELATED WORK

Song et al. [1] first introduced the notion of searchable encryption. They proposed that each word of a document be encrypted with a special encryption construct and then search for documents that contain a single specified keyword. Its search time is related linearly with the length of the file collection, which means the server must scan all of the data points to find a match. Subsequently, Goh et al. [18] proposed the use of indexes and developed a per-file index design via a Bloom filter. In addition, they also proposed a security definition to formalize the security requirements of SSE. Similarly, Chang et al. [16] introduced a simulation-based security definition that was slightly stronger than the definition in [18]. However, none of the above definition considered adaptive adversaries, which could generate the queries according to the outcomes of previous queries. The shortcomings of [16] and [18] were addressed by Curtmola et al. [19], who presented an adaptive security definition for searchable encryption schemes. They improved the security notions known as SSE and also provided a protocol that was compatible with their definitions. Their scheme was the first to use inverted indexes. Thus, compared with the previous schemes, their scheme provided better security and efficiency.

However, none of the schemes presented to date enable similarity matching; they only provide exact matching in the context of a keyword search. Hence, such schemes are subject to typographical errors, so these schemes are not applicable for many real-world search scenarios. Li et al. [20] proposed a wildcard-based, fuzzy-keyword search scheme over encrypted data. Although the fuzzy scheme tolerates errors to some extent, it is only applicable to strings under edit distance. Therefore, fuzzy sets may become too large if there are long words that necessitate issuing large trapdoors. Another similarity-matching technique for encrypted data was proposed in [21]. Their approach is applicable to approximate string search under hamming distance. In contrast to our study, their approach assumes the categorical division of the data sources. For instance, documents are divided into fields (e.g., email to and email from fields) in such a way that each field has a single value. Wang et al. [22] used a suppressing technique to build a storage-efficient similarity keyword set from a given document collection, and they used the edit distance as the similarity metric. Wang et al. [23] proposed a multi-keyword, fuzzy search scheme based on LSH. Sun et al. [24] designed a search index based on the frequency of terms and the vector space model with a cosine similarity measure that could support multi-keyword searches and ranking of the search results. In addition to a similar search, Cash et al. [25] designed a protocol that supports conjunctive searches and general Boolean queries over encrypted data.

Although the SSE schemes mentioned above can support keyword-based searches effectively, they do not enable similarity searches over encrypted high-dimensional data, such as images, multimedia data, and web pages. SSE is applicable for any form of retrieval of private information based on keywords. Thus, whether built on LSH [26] or other embedding techniques, similarity search over high-dimensional data is transformed to keyword searches. Kuzu et al. [7] developed an encrypted index based on an LSH-based inverted index. Each distinct LSH hash value is associated with an $n$-bit vector, where $n$ is the total number of records in a dataset, and each bit indicates a matched record. Due to the rapid development of LSH in recent years [27]–[29], many methods of combining LSH and SSE have emerged and they have produced advancements from different perspectives, e.g., index structure [9], search efficiency [17], and security [30]. Their improvements have made different contributions.

However, the above schemes overlooked the data distributions that occur in similarity searches over encrypted, high-dimensional data. In practice, data distributions often are non-uniform and frequently very skewed, and it usually

TABLE 1: Method comparison

| Method | Query quality on non-uniform datasets | Against CQA2 | Hiding search distribution |
|---|---|---|---|
| Chang et al. [16] | General quality | × | × |
| Kuzu et al. [7] | General quality | √ | × |
| Wang et al. [17] | General quality | √ | × |
| SEND | High quality | √ | √ |

causes the structures of hash tables to be unbalanced, which impairs the performance of the solution. Therefore, it is necessary to take the distribution of the data into consideration. A searchable encryption scheme that considers the distribution of the data will make a significant difference to many actual datasets.

Security is another important requirement for SSE. In the field of keyword searches of SSE, there are many research works on improving security [31]–[33]. However, in the field of high-dimensional searches, the method of improving security is inadequate. In most of the previous schemes, one of their common security problems is evident, i.e., it is easy for the cloud server to get the query distribution. According to the query distribution, the cloud server can use statistical methods to infer large amounts of sensitive information [15]. If some additional background information is known, the consequences will be disastrous. In consideration of our higher requirement for protecting privacy and making information secure, a solution that can hide the query distribution to provide higher security is an urgent need.

We have combined the two problems mentioned above in the field of searchable encryption, that is, the impact of data distribution on performance and the leakage of query distribution. We propose a solution that builds the index based on the data distribution and can hide the query distribution. TABLE 1 lists the method comparison between existing schemes and our scheme.

## 3 Preliminaries

### 3.1 Locality Sensitive Hashing

Locality-sensitive hashing (LSH) [26] is an efficient algorithm to solve the problem of approximate nearest neighbors in high-dimensional space. The general idea of LSH is that objects that are within a given distance will be hashed to the same value with high probability. LSH uses a set of hash functions to map objects into several buckets such that there is a high probability that similar objects will share the same bucket, whereas dissimilar objects do not. We denote the sphere centered at point $q$ with a radius $r$ by $B(q, r)$. Then, for a $d$-dimensional metric space $R^d$ of the data points and a result set $U$, the locality-sensitive function family that LSH uses is defined as shown below:

***Definition 3.1 (LSH Family $\mathcal{H}$).*** *A family $\mathcal{H} = \{h : R^d \to U\}$ of functions is called $(r, cr, p_1, p_2)$-sensitive if $\forall p, q \in R^d$:*

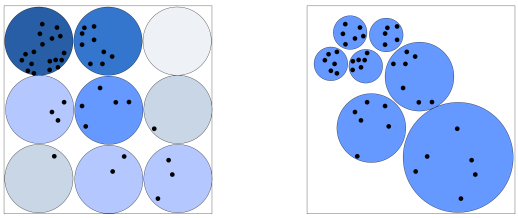
- *if $p \in B(q, r)$, then $Pr_{\mathcal{H}}(h(p) = h(q)) \geq p_1$;*
- *if $p \notin B(q, cr)$, then $Pr_{\mathcal{H}}(h(p) = h(q)) \leq p_2$.*

Note that we should at least ensure that $c > 1$ and $p_1 > p_2$ to use LSH for the similarity search. To reduce the false positive rate, we enlarge the gap between the two probabilities, i.e., making $p_1 \gg p_2$.

There are several distance functions of LSH, e.g., Manhattan distance, Euclidean distance, Jaccard distance, and Cosine distance. In this paper, we choose to use $l_2$ norm, i.e., the Euclidean distance. Specifically, each hash function is defined by:

$$h(q) = \lfloor \frac{a \cdot q + b}{r} \rfloor \tag{1}$$

where $a$ is a $d$-dimensional vector with each dimension chosen independently from a 2-stable distribution: normal distribution $g(x) = \frac{1}{\sqrt{2\pi}} e^{-x^2/2}$ and $b$ is chosen uniformly from $[0, r]$, where $r$ is the length range of $q$ for searching for similar points. From a geometric perspective, we can understand that a data point $q$ in $R^d$ is projected onto a random line $a$ which is segmented into intervals of equal width with length $r$.



(a) LSH (b) SH(selective hashing)

Fig. 1: Range of searching similarity points in LSH and SH.

Note that the LSH family aims at searching the similar objects of $q$ within a fixed radius, $r$. However, this design causes the structures of the hash tables to be unbalanced when the dataset is not uniform. Fig. 1(a) shows an example of the hashing results of LSH. On the indexing level, we see that the hashing is unbalanced, i.e., some buckets are empty, but a few buckets contain too many points. This means that a large number of points are returned if the query point falls in the dense regions, resulting in unsatisfactory precision and reduced efficiency. If the query point falls into the sparse regions, only a very few points, or even none, are returned, resulting in low recall. Consequently, this causes the performance of search quality to degrade when the distribution of the dataset is non-uniform. As shown in Fig. 2, LSH achieves good search quality when it is used for a uniform dataset. However, both the recall and precision obviously are decreased when the dataset is non-uniform.

### 3.2 Selective Hashing

To make the similarity search effective on a non-uniform dataset, it is necessary to take the data distribution, i.e.,

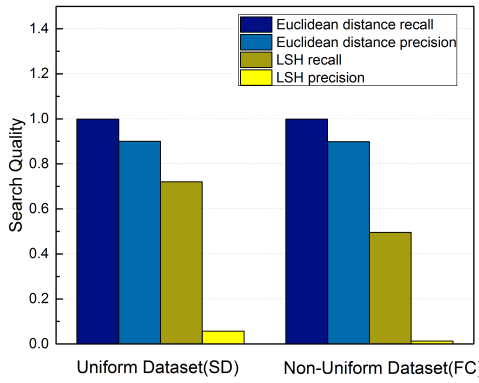



Fig. 2: Effect of the data distribution on the quality of the LSH search.

data density, into account when building the index. Some methods were proposed for estimating the density of multi-dimensional data in [34]–[36]. In this paper, we use selective hashing [36] to build the index.

In selective hashing (SH), instead of having a fixed radius, each object has its own search radius based on the local density near it as shown in Fig. 1(b). Thus, each data object has a radius with an appropriate size. The denser the region is, the smaller radius the data objects in the region will have. Conversely, the sparser the region is, the larger the radius the data objects in the region will have. Thus, the number of similar objects is almost as same for each data object. In this way, the problem of poor performance in traditional LSH is solved when the data are very skewed and SH achieves high-quality searches in terms of recall and precision. Due to the excellent performance of selective hashing over a non-uniform dataset, we decide to apply selective hashing to construct our scheme. The key to selecting an appropriate radius for each data point is the density of the data around that data point. This is formalized as:

***Definition 3.2 (Data Density*** $\rho_D(q,r)$***).*** *Let $r$ be the optimal radius of the data object $q$. Then, the data density of $q$ in dataset $D$ is the number of points in $B(q,r)$ that are defined by $\rho_D(q,r) = |\{o|o \in D, o \in B(q,r)\}|$.*

The estimate of the number of points in $B(q,r)$ is computed by the collisions in the LSH tables. A detailed description in our constructions is provided in Section 5.

Since we build indexes based on data distribution, it is necessary to give a definition of data distribution uniformity. The definition of data distribution uniformity is as follows.

***Definition 3.3 (Uniformity of Data Distribution*** $UDD(D)$***).*** *Let $r_i$ be the optimal radius of the data object $q_i$ in $D$. Then, the uniformity of data distribution of $D$ is the variance of the optimal radius for all points in $D$ that is defined by $UDD = \frac{\sum_1^n (\bar{r}-r_i)^2}{n}$ where $\bar{r} = \frac{\sum_1^n r_i}{n}$.*

Based on the above definition, we can say that the smaller the $UDD$ value, the more uniform the dataset distribution.

### 3.3 Inverted Index

An inverted index is an index data structure that stores a mapping from content, such as words or numbers, to its locations in a database file, in a document, or in a set of documents. Fig. 3 shows the structure of an inverted index. The documents containing keyword1 are file2 and file3 and so forth.

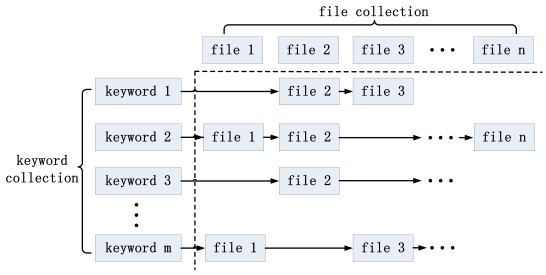



Fig. 3: Inverted index.

### 3.4 Cryptographic Primitive

A private-key encryption scheme is a set of three polynomial-time algorithms $Gen, Enc, Dec$. The probabilistic key generation algorithm $Gen$ takes a security parameter $\lambda$ to return a secret key $sk$, that satisfies $|sk| > \lambda$. The probabilistic encryption algorithm $Enc$ takes a key $sk$ and a plaintext $m \in \{0,1\}^*$ to return a ciphertext $c \in \{0,1\}^*$. The deterministic decryption algorithm $Dec$, takes $sk$ and $c \in \{0,1\}^*$ to return $m \in \{0,1\}^*$. In our construction, we use this encryption scheme to encrypt the dataset and index content.

In addition to encryption scheme, we also make use of pseudo-random permutations (PRP) in SEND. PRP is a kind of pseudo-random functions (PRF). If $F$: $\{0,1\}^\lambda \times \{0,1\}^{poly(\lambda)} \rightarrow \{0,1\}^{poly(\lambda)}$ is a PRF, it is a polynomial-time computable function that cannot be distinguished from a random function by any polynomial time adversary [37]. In particular, the function is bijective for PRP. For higher security, the buckets of hash tables are permuted through a pseudo random permutation in our construction.

## 4 SYSTEM MODEL AND SECURITY DEFINITION

### 4.1 Similarity Search over Encrypted High-dimensional Data

Fig. 4 shows that we consider our cloud data hosting storage system that involved four entities, i.e., the owner of the data, the user of the data, the cloud server $A$ and the cloud server $B$. The owner and user of the data are also known as clients. The owner of the data encrypts the data locally before outsourcing them. The collection of $n$ sensitive, $d$-demensional data is denoted by $D = (D_1, D_2, ..., D_n)$. The corresponding encrypted form of $D$ is $C = (C_1, C_2, ..., C_n)$. To keep the capability of searching over $C$ for effective retrieval, the data owner builds an encrypted data index $\mathcal{I}_D$ and a random index $\mathcal{I}_R$ based on $D$. Then the owner outsources $\gamma = (\mathcal{I}_D, C)$ to the cloud server $A$ and outsources $\mathcal{I}_R$ to the server $B$.

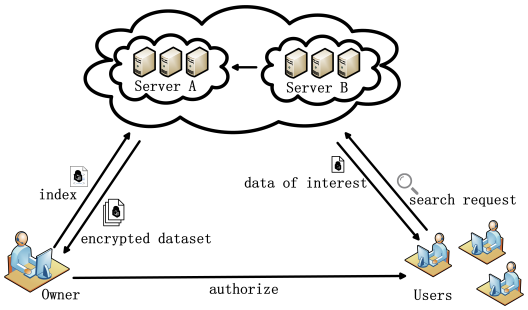



Fig. 4: Architecture of our scheme.

We assume that the authorization between the data owner and the data users is done appropriately. To search a similar collection of data for a given search object $q$, the data user generates a search token $T_q$ for $q$, a search request in an encrypted form, and sends it to the cloud server $B$. Upon receiving the search request $T_q$, the cloud server $B$ randomizes $T_q$ based on $\mathcal{I}_R$ to obtain a randomized trapdoor $T_q^{'}$ and then the server $B$ sends $T_q^{'}$ to the server $A$. The server $A$ is responsible for searching over the index $\mathcal{I}_D$, and returning $C_q$, a sequence of encrypted data the corresponding plaintext data of which have characteristics similar to $q$.

The core functionalities are defined below.

***Definition 4.1.*** *A scheme that supports similarity searches over an encrypted dataset consists of six polynomial-time algorithms as follows:*

- $sk \leftarrow Gen(1^{\lambda})$ *is a probabilistic key generation algorithm run by the client. It takes a security parameter* $\lambda$*, as input and outputs the secret key* $sk$*.*
- $(\mathcal{I}, C) \leftarrow Enc(sk, D)$ *is a probabilistic algorithm run by the client. It takes as input a secret key* $sk$*, and a collection of data,* $D$*, and outputs an encrypted index* $\mathcal{I}$*, and an encrypted data collection* $C$*.*
- $T_q \leftarrow SrchToken(sk, q)$ *is a (possibly probabilistic) algorithm run by the client. It takes as input a secret key* $sk$*, and a search object* $q$*, and outputs a search token* $T_q$*.*
- $T_q^{'} \leftarrow RdmToken(\mathcal{I}_R, T_q)$ *is a probabilistic algorithm run by the server* B*. It takes as input a random index* $\mathcal{I}_R$*, and a search request* $T_q$*, and outputs a randomized trapdoor*$T_q^{'}$*.*
- $C_q \leftarrow Search(\mathcal{I}_D, T_q^{'})$ *is a deterministic algorithm run by the cloud server* A*. It takes as input an encrypted index* $\mathcal{I}$*, and a search token* $T_q$*, and outputs a sequence of encrypted data* $C_q$*, where* $C_q \subseteq C$*.*
- $D_q \leftarrow Dec(sk, C_q)$ *is a deterministic algorithm run by the client. It takes as input a secret key* $sk$*, and a sequence of encrypted data* $C_q$*, and outputs a sequence of plaintext data* $D_q$*.*

It should be noted that the data we use is capable of being clustered. Beyer et al. [38] show that under a broad set of conditions, the distance to the nearest data point approaches the distance to the farthest data point with the increase in dimensionality. That is to say, in the data set that cannot be clustered, as the data dimension increases, the contrast in distances to different data points becomes very obvious. Therefore, we accept the suggestion made in [38], and we make the encrypted high-dimensional data similarity search scheme premise to search on the data that is clustered.

### 4.2 Security Definition

Basically, we consider a server to be *honest-but-curious*, which is consistent with most of the previous searchable encryption schemes. We assume that the cloud server acts in an honest manner and correctly follows the specified protocol. However, at the same time, it is curious to infer and analyze the message received during the protocol. In addition, in the multi-server setting, we assume that both servers are *honest-but-curious* and do not collaborate with each other, which is a common threat model for multi-party protocols [7], [39]–[41]. and is also consistent with the common sense that cloud service providers with conflicts of interest do not cooperate in practice.

Our scheme follows the widely-accepted security notion of SSE stated in [19], [42]. Also, we introduce one more definition, i.e., the definition of search distribution. Intuitively, the similarity search over encrypted, high-dimensional data should provide certain security guarantees, i.e., 1) the server cannot learn any sensitive information about the data from the encrypted index and 2) the server can only learn what is allowed to be leaked by the client. Like almost all prior practical SSE schemes, what is allowed to be leaked are the search pattern and the access pattern. Explicitly, the search pattern indicates whether or not the same search has been performed in the past. The access pattern includes the results of the queries. Their formal definitions are provided below:

***Definition 4.2 (Search Pattern $\pi$).*** *Let* $q_1, q_2, ..., q_n$ *be the set of* $n$ *consecutive queries. Then, the search pattern* $\pi$ *is defined by a binary matrix* $\pi(i, j) = 1$ *if* $q_i = q_j$ *and* $\pi(i, j) = 0$ *otherwise.*

***Definition 4.3 (Access Pattern $A_q$).*** *Let* $id(c_q)$ *be the identifier of an encrypted data item that corresponds to the search object* $q$*. Then, the access pattern is a collection defined by* $A_q(q) = id(c_q)$*.*

In our scheme, we try to achieve higher security. Thus, we introduce a new definition of search distribution to describe the server's analysis for a search pattern. Intuitively, it indicates the number of distinct queries that occurred. Specifically, it is formalized as shown below.

***Definition 4.4 (Search Distribution $\mathcal{X}$).*** *Let* $n_t$ *be the number of distinct search objects, and let* $N_i$ *be the number of occurrences of the search object* $q_i$*. Then, the search distribution is defined by* $\mathcal{X} = (N_1, N_2, ..., N_{n_t})$*.*

The search distribution that the server learned may lead to statistical attacks [15], especially when the adversary has partial or full knowledge of the queries. In this paper, we use a random index to flatten the search distribution. In this

way, the numbers of the distinct search objects are all the same in the view of the server.

A crucial issue with respect to the security of SSE is whether the scheme is secure against adaptive chosen query attacks (CQA2) or only against non-adaptive chosen query attacks (CQA1). The former guarantees security even when the clients queries are based on the encrypted index and the results of previous queries. The latter only guarantees security if the clients queries are independent of the index and the previous results. Our proposed scheme is secure against CQA2, and, in section 5, we provide proof based on the simulation-based model [19] and leakage functions [43]. The formal definition is provided below:

***Definition 4.5 (CQA2-security).*** *Let* $\Omega = (Gen, Enc, SrchToken, RdmToken, Search, Dec)$ *be SEND for secure similarity searches, and let* $\mathcal{L}_1$, $\mathcal{L}_2$ *and* $\mathcal{L}_3$ *be the leakage functions for the view of the encrypted index, the search pattern, and the access pattern, respectively. Given an adversary,* $\mathcal{A}$*, and a simulator,* $\mathcal{S}$*, the following probabilistic games, i.e.,* $Real_{\mathcal{A}}(\lambda)$ *and* $Ideal_{\mathcal{A},\mathcal{S}}(\lambda)$*, are defined:*

*$Real_{\mathcal{A}}(\lambda)$: A challenger calls $Gen(a^{\lambda})$ to output a key $sk$. $\mathcal{A}$ selects $Dand$ asks the challenger to build $\mathcal{I}$ via $Enc$. Then, $\mathcal{A}$ adaptively makes a polynomial number of $Search$ queries and asks for the trapdoor $T_q$ of each query $q$ from the challenger. Then, $\mathcal{A}$returns a bit as the games output.*

*$Ideal_{\mathcal{A},\mathcal{S}}(\lambda)$: $\mathcal{A}$ selects $D$, and $\mathcal{S}$ generates $\widetilde{\mathcal{I}}$ based on $\mathcal{L}_1(D)$. Then, $\mathcal{A}$adaptively performs a polynomial number of queries. From $\mathcal{L}_2(D)$ and $\mathcal{L}_3(D)$ of each query $q$, $\mathcal{S}$ generates the corresponding $\widetilde{T_q}$. Then, $\mathcal{A}$ returns a bit as the games output.*

*Our proposed scheme $\Omega$ is secure against CQA2, if for all probabilistic polynomial time adversaries $\mathcal{A}$, there exists a probabilistic polynomial time simulator $\mathcal{S}$, such that:*

$$Pr[Real_{\mathcal{A}}(\lambda) = 1] - Pr[Ideal_{\mathcal{A},\mathcal{S}}(\lambda) = 1] \leq \epsilon(\lambda) \quad (2)$$

*where $\epsilon(\lambda)$ is a negligible function in $\lambda$.*

# 5 SEND: SECURE SIMILARITY SEARCH OVER ENCRYPTED NON-UNIFORM DATASETS

In this section, we present our secure scheme SEND for similarity search over an encrypted, non-uniform dataset. The remainder of this subsection is organized as follows. We first introduce the overview of SEND. Then we introduce the details of SEND. Furthermore, we describe SEND in two phases in the details of SEND. The two phases are index construction phase and similarity search phase, respectively. More specifically, the index construction phase consists of six sub-phases and the similarity search phase consists of four sub-phases.

## 5.1 Overview of SEND

Fig. 5 shows our scheme workflow. SEND can be divided into two phases: index construction phase and similarity search phase.

In index construction phase, firstly, ① the data owner encrypts the dataset using symmetric encryption before outsourcing them to the cloud. ② Then the data owner selects the optimal radius for each data point based on selective hashing. Subsequently, ③ the data owner build a data index $\mathcal{I}_D$ and a random index $\mathcal{I}_R$ based on the optimal radius of each point. Finally, ④ the data owner stores encrypted dataset and $\mathcal{I}_D$ on server $A$, ⑤ and stores $\mathcal{I}_R$ on server $B$.

In similarity search phase, when a data user wants to search data points that are similar to data point $q$, ⑥ he or she sends a trapdoor $T_q$ to server $B$. ⑦ Server $B$ randomizes the trapdoor $T_q$ by adding a random number to it, resulting in random trapdoor $T_q^{'}$ and then server $B$ sends $T_q^{'}$ to server $A$. ⑧ Server $A$ finds the corresponding buckets in $\mathcal{I}_D$ according to $T_q^{'}$ and ⑨ returns the data points stored in those buckets to the user after making some modifications to the content of those buckets.

## 5.2 Details of SEND

### 5.2.1 Index construction phase

**Step-1. Index Initialization**

To address the poor performance problem of LSH on non-uniform dataset, we use LSH as the basis for the rest of our construction and then use selective hashing, which is an improved algorithm based on the traditional LSH. In this paper, the index is an inverted index. We consider the LSH hash values as keywords and consider the $dataIDs$ as files. The hash family is weak, so we use concatenation to generate stronger hash functions. We use $L$ concatenation functions to construct our scheme, and each concatenation function consists of $M$ random functions. The value of a concatenation function is XOR of its $M$ hash values. The $L$ concatenation functions are denoted by $g_1, g_2, ..., g_L$, and each concatenation function, $g_i \in g_1, g_2, ..., g_L$, is denoted by $g_i = (h_{i_1} \oplus h_{i_2} \oplus ... \oplus h_{i_M})$. Each concatenation function corresponds to a hash table. Each data point is associated with $L$ keywords for concatenation functions. As a result, $L$ hash tables are built, where each LSH hash value links to a list of identifiers of collided data points.

Random index $\mathcal{I}_R$ for randomizing the query trapdoor, data index $\mathcal{I}_D$ for search similar data objects according randomized query trapdoor and a temporary index $\mathcal{I}_{temp}$ for radius selection have the same structure containing $L \times |R|$ hash tables, as shown in Fig. 6. Each hash table has $b$ buckets. We consider the hash tables with the same radius as a $subindex$.

The data are stored in $\mathcal{I}_R$ as a key-value pair $< x_{(j,z)}, numList_{(j,z)} >$, where $x_{(j,z)}$ is the bucket position. $numList_{(j,z)}$ denotes a set of random numbers and it is an empty set when initialized. The data are stored in $\mathcal{I}_D$ as a key-value pair $< x_{(j,z)}^{'}, dataIDList_{(j,z)} >$, where $x_{(j,z)}^{'} = x_{(j,z)} + sum(numList_{(j,z)})$. The term $sum(\cdot)$ is a function that sums all the numbers in a set, and $numList_{(j,z)}$ can be found in $\mathcal{I}_R$. The term $dataIDList_{(j,z)}$ refers to a set of data identifiers.

**Step-2. Parameter Initialization**

Given a security parameter, $\lambda$, select the following parameters uniformly at random from the corresponding domains:

- $k_1 \leftarrow SSE.Gen(1^{\lambda})$ a symmetric key for index symmetric encryption;
- a PRP key $k_2 \leftarrow \{0,1\}^{\lambda}$ for $P_{k_2}(\cdot)$;

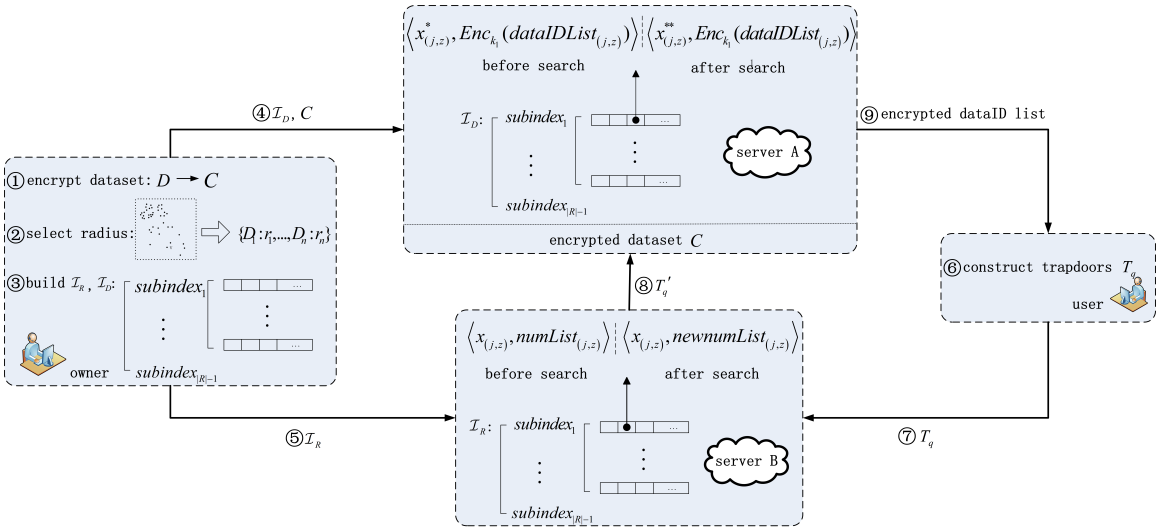



Fig. 5: Workflow of our scheme.

- a symmetric key $k_3 \leftarrow SSE.Gen(1^\lambda)$ for data symmetric encryption;
- a radius candidate set $R = \{r, cr, c^2r, ..., c^{|R|-1}r\}$, where $r$ is the smallest radius, $c$ is the increment ratio, and $R$ is the length of $R$;
- a vector set for building the index based on LSH, $A = \{a_{11}, a_{12}, ..., a_{1M}, ...a_{L1}, a_{L2}, ..., a_{LM}\}$.

The output of this algorithm is $sk = (k_1, k_2, k_3, R, A)$.

**Algorithm 1** Index Construction

**Input:** $D$, $\lambda$, $g$, $\mathcal{I}_{temp}$(index for radius selection)
**Output:** $\mathcal{I}_R, \mathcal{I}_D$
1: $\{k_1, k_2, k_3\} \rightarrow Gen\left(1^\lambda\right)$
2: **for** each $D_i$ in $D$ **do**
3: exract feature vector of $D_i$ then get $p_i$
4: compute $x_{(1,r)}\left(p_i\right), ..., x_{\left(L,c^{|R|-1}\right)r}\left(p_i\right)$
5: insert $p_i$ into $\mathcal{I}_{temp}$
6: **end for**
7: **for** each $D_i$ in $D$ **do**
8: count the number of points in the buckets $x_{(1,r)}\left(p_i\right)$,..., $x_{\left(L,c^{|R|-1}\right)r}\left(p_i\right)$ of $\mathcal{I}_{temp}$
9: select optimal radius $r_i$ for $D_i$
10: **end for**
11: **for** each $D_i$ in $D$ **do**
12: compute $x_{(1,r_i)}\left(p_i\right), ..., x_{(L,r_i)}\left(p_i\right)$
13: generate a random number $\delta$
14: **for** $j$ in $[1, L]$ **do**
15: insert $\delta$ into $numList_{(j,r_i)}$
16: $x'_{(j,r_i))} = x_{(j,r_i)} + sum(numList_{(j,r_i)}))$
17: insert $dataID_i$ into $DataIDList_{(j,r_i)}$
18: $< x_{(j,r_i)}, numList_{(j,r_i)} > \rightarrow \mathcal{I}_R$
19: $< x_{(j,r_i)}, DataIDList_{(j,r_i)} > \rightarrow \mathcal{I}_D$
20: **end for**
21: **end for**
22: $\mathcal{I}_D = Enc(k_1, \mathcal{I}_D)$
23: **return** $\mathcal{I}_R, \mathcal{I}_D$

**Step-3. Construct The Concatenation Functions**

Construct the functions $(g_{(1,c^kr)}, g_{(2,c^kr)}, ..., g_{(L,c^kr)})$, where

$$g_{(j,c^kr)}(q) = h_{j_1}(q) \oplus h_{j_2}(q) \oplus ... \oplus h_{j_M}(q) = \frac{a_{j_1} \cdot q}{c^k r} \oplus \frac{a_{j_2} \cdot q}{c^k r} \oplus ... \oplus \frac{a_{j_M} \cdot q}{c^k r} \quad (3)$$

for $j \in [1, L], k \in [0, |R| - 1]$.

Compute the $L \times |R|$ hash bucket positions of $p_i$, $(x_{(1,r)}, ..., x_{(L,r)}, ..., x_{(1,c^{|R|-1}r)}, .., x_{(L,c^{|R|-1}r)})$, where

$$x_{(j,z)} = g_{(j,z)}(p_i) \bmod b, j \in [1, L], z \in R. \quad (4)$$

The data owner uses relevant technique to exract feature vector $p_i$ for each data object $D_i$ (line 3 in Algorithm 1). Next, $p_i$ is inserted into $\mathcal{I}_{temp}$. The position of the inserted bucket is calculated according to Equation (3) and (4) (line 4-5 in Algorithm 1).

**Step-4. Radius Selection**

We set a threshold $\alpha$ for the number of points in a bucket. For a data object, we use $v \in \{0, 1\}^L$ to represent the relationship between $\alpha$ and the number of points in $L$ buckets of a $subindex$. For $v$, we refer its $j^{th}$ bit as $v[j]$. Set a threshold $\beta$ for the sum of each dimension of $v$. For each $D_i \in D$: Count the number of points hashed in $(x_{(1,r)}, ..., x_{(L,r)}, ..., x_{(1,c^{|R|-1}r)}, .., x_{(L,c^{|R|-1}r)})$. For $subindex_k$, if the number of points in $x_{(j,c^kr)}$ is greater than $\alpha$, $v[j] = 1$, otherwise, it is 0. Compute $s = v[1] + v[2] + ... + v[L]$. Find the $subindex$ that satisfies $s \geq \beta$ and has the smallest radius; set its radius to the optimal radius of $D_i$, denoted by $r_i$ (line 7-10 in Algorithm 1). Fig. 7 show an example, we assume that it shows the hash result of point $D_1$. We use a shaded bucket to represent the bucket where the number of points exceeds $\alpha$. Then,

- for $subindex_0$, $v = \{0, 0, 1\}$, $s = 1$;
- for $subindex_1$, $v = \{1, 1, 1\}$, $s = 3$;
- for $subindex_2$, $v = \{0, 1, 1\}$, $s = 2$.

If $\beta = 2$, we set the radius of $subindex_1$, i.e., $cr$, to be the optimal radius of $D_1$.

**Step-5. Data Insertion**

For $D_i \in D$, the SEND executes as follows.

1) Compute $L$ hash bucket positions of $p_i$ in $\mathcal{I}_R$ and $\mathcal{I}_D$, $(x_{(1,r_i)}, x_{(2,r_i)}, ..., x_{(L,r_i)})$, where $r_i$ is the optimal radius of $p_i$ (line 12 in Algorithm 1).

2) Insert a random number into $numList_{j,(r_i)}$ in $\mathcal{I}_R$, where $j$ in $[1, L]$.

3) Insert $dataID_i$, the identifier of $D_i$, into $DataIDList_{(j,r_i)}$ in $\mathcal{I}_D$, where $j$ in $[1, L]$.

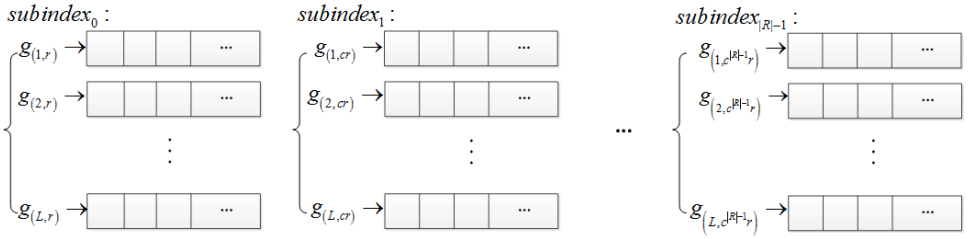



Fig. 6: Hash tables for $\mathcal{I}_R$ and $\mathcal{I}_D$.

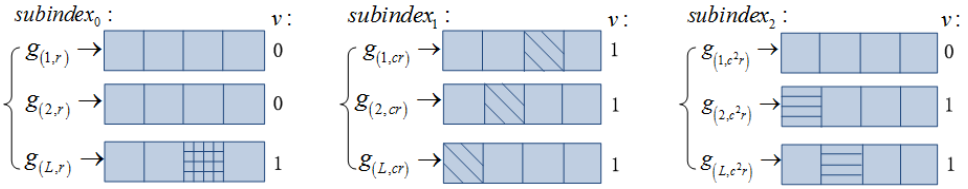



Fig. 7: Example of selecting the radius.

4) Update $L$ hash bucket positions of $p_i$ in $\mathcal{I}_R$ to $< x_{(j,r_i)}, numList_{(j,r_i)} >$, where $j$ in $[1, L]$.

5) Update $L$ hash bucket positions of $p_i$ in $\mathcal{I}_D$ to $< x^{'}_{(j,r_i)}, DataIDList_{(j,r_i)} >$, where $j$ in $[1, L]$.

**Step-6. Encryption**

1) Fill the empty buckets with dummy padding.

2) Permute $\mathcal{I}_D$ according to the mapping $x_{(j,z)}$ to the randomized position $P_{k_2}(x_{(j,z)})$ for each position $x_{(j,z)}$.

3) Encrypt each bucket in $\mathcal{I}_D$ by a symmetric encryption algorithm in the form of $< Enc(k_1, x^{'}_{(j,z)}), Enc(k_1, dataIDList_{(j,z)}) >$.

4) Encrypt each data object $D_i \in D$ by a symmetric encryption algorithm in the form of $C_i = Enc(k_3, D_i)$.

5) Outsource $\mathcal{I}_D$ and $C$ to Server $A$. Outsource $\mathcal{I}_R$ to Server $B$.

### 5.2.2 Similarity search phase

**Step-1. [The User Side] Trapdoor Generation (line 1-4 in Algorithm 2)**

1) Extract the $d$-dimensional feature vector, $q$.

2) Compute the trapdoor $T_q$ by

$$T_q = \begin{pmatrix} x_{(1,r)} & x_{(1,cr)} & \dots & x_{(1,c^{|R|-1}r)} \\ x_{(2,r)} & x_{(2,cr)} & \dots & x_{(2,c^{|R|-1}r)} \\ \dots & \dots & \dots & \dots \\ x_{(L,r)} & x_{(L,cr)} & \dots & x_{(L,c^{|R|-1}r)} \end{pmatrix} \quad (5)$$

3) Send $T_q$ to server $B$.

**Step-2. [The Server *B* Side] Trapdoor Randomization By $\mathcal{I}_R$ (line 5-13 in Algorithm 2)**

For each $x_{(j,z)}$ in $T_q$,

1) Compute

$$x^{'}_{(j,z)} = x_{(j,z)} + sum(numList_{(j,z)}); \quad (6)$$

$$x^{*}_{(j,z)} = Enc(k_1, x^{'}_{(j,z)}). \quad (7)$$

2) Insert a new random number into $numList_{(j,z)}$ to obtain the $newnumList_{(j,z)}$. This guarantees that the next randomization of the same trapdoor will be distinct from the previous ones.

3) Compute

$$x^{**}_{(j,z)} = Enc(k_1, x_{(j,z)} + sum(newnumList_{(j,z)})). \quad (8)$$

4) Generate the randomized trapdoor, $T^{'}_q$, by:

$$T^{'}_q = \begin{pmatrix} (x^{*}_{(1,r)}, x^{**}_{(1,r)}) & \dots & (x^{*}_{(1,c^{|R|-1}r)}, x^{**}_{(1,c^{|R|-1}r)}) \\ (x^{*}_{(2,r)}, x^{**}_{(2,r)}) & \dots & (x^{*}_{(2,c^{|R|-1}r)}, x^{**}_{(2,c^{|R|-1}r)}) \\ \dots & \dots & \dots \\ (x^{*}_{(L,r)}, x^{**}_{(L,r)}) & \dots & (x^{*}_{(L,c^{|R|-1}r)}, x^{**}_{(L,c^{|R|-1}r)}) \end{pmatrix} \quad (9)$$

5) Send $T^{'}_q$ to server $A$.

**Step-3. [The Server *A* Side] Search Over $\mathcal{I}_D$ (line 16-23 in Algorithm 2)**

1) Return the value, i.e., $DataIDList_{(j,z)}$, that corresponds to the key, $x^{*}_{(j,z)}$.

2) For the buckets that just have been accessed, modify their content from $< x_{(j,z)^*}, Enc(k_1, dataIDList_{(j,z)}) >$to $< x_{(j,z)^{**}}, Enc(k_1, dataIDList_{(j,z)}) >$.

3) Permute the buckets of $\mathcal{I}_D$ by $P_{k_1}(\cdot)$ so that server $A$ cannot learn the frequency of each bucket that is being accessed. In this way, SEND provides higher security.

**Step-4. [The User Side] Decryption**

For each $C_i \in C$, decrypt $C_i$ by a symmetric encryption algorithm in the form of $D_i = Dec_{k_3}(C_i)$.

In the previous schemes with different radii, each data object is usually stored in every $subindex$, so the problem of overlapping between the points returned by each $subindex$ is very serious when searching for similar data. Obviously, these overlapping points are the points with higher similarity. However, in SSE, the index for the similarity search must be encrypted to preserve privacy. As a result, we cannot distinguish between points in the cloud server that overlap to a high extent. If all of the points that are found are returned to the data user and then the user performs the filtering job locally, it will make the users work more time-consuming and troublesome.

Although there are many hash tables that correspond to different radii in our index, each data object is only stored in the $subindex$ with its optimal radius. When retrieving similar data of a data object, we still check for each sub

index, but there is no overlap between the points returned by each sub-index, and the $subindex$ with the optimal radius of this data object guarantees the data points with higher degree of similarity will be returned to the client. In fact, most of the points returned are the ones with a higher similarity. In this way, we can make the users job easier and also ensure the high precision of the returned data.

**Algorithm 2** Similarity Search

**Input:** $Q, \mathcal{I}_R, \mathcal{I}_D$
**Output:** $resultList$
**Client:**
exact feature vector of $Q$ then get $q$
compute $T_q$=($x_{(1,r)}(q), ..., x_{(L,c^{|R|-1})_r}(q)$)
send $T_q$ to Server $B$
**Server *B*:**
**for** each $x_{(j,z)}$ in $T_q$ **do**
  $x^{'}_{(j,z)} = x_{(j,z)} + sum(numList_{(j,z)})$
  $x^{*}_{(j,z)} = Enc(k_1, x^{'}_{(j,z)})$
  insert a random number into $numList_{(j,z)}$ to get $newnumList_{(j,z)}$
  $x^{**}_{(j,z)} = Enc(k_1, x_{(j,z)} + sum(newnumList_{(j,z)})$
  $numList_{(j,z)} = newnumList_{(j,z)}$
**end for**
$T^{'}_q = (< x^{*}_{(1,r)}, x^{**}_{(1,r)} >, ..., < x^{*}_{(L,c^{|R|-1}r)}, x^{**}_{(L,c^{|R|-1}r)} >$
send $T^{'}_q$ to Server $A$
**Server *A*:**
**for** each bucket in $\mathcal{I}_D$ **do**
  **if** $x^{*}_{(j,z)} \in T^{'}_q$ **then**
    $resultList.insert(dataIDList_{(j,z)})$
    modify the bucket to $< x^{**}_{(j,z)}, dataIDList_{(j,z)} >$
  **end if**
**end for**
permute $\mathcal{I}_D$
**return** $resultList$ to Client

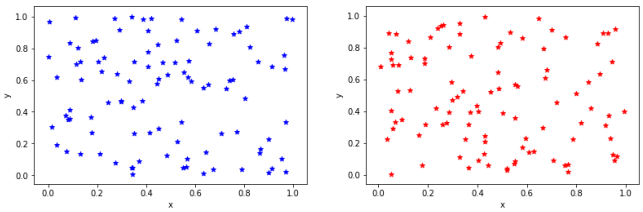
## 6 SECURITY ANALYSIS

In this section, we analyze the security of SEND under the security framework of SSE. We define the leakage functions for the index, search pattern, and access pattern which are the information that the server can learn from our scheme. In SEND, the main opponent is server $A$ because all of our data are stored on it. Then, based on the defined leakage functions, we can prove that SEND is secure against adaptive, chosen-query attacks (CQA2).

It should be noted that our scheme has better search quality on non-uniform datasets, but in the security analysis section, we do not need to pay attention to the data distribution in the dataset. As shown in Fig. 8, the encrypted distribution of a uniform dataset Fig. 8(a) and the encrypted distribution Fig. 8(b) of a non-uniform data set cannot be distinguished. So in this section, we don't need to pay attention to the distribution of data in the dataset.

***Definition 6.1 (Leakage Function $\mathcal{L}_1$ for Index).*** *Given an encrypted index $\mathcal{I}$, $\mathcal{L}_1 = (L \times |R|, b, < |u|, |v| >)$, $L \times |R|$ denotes the number of hash tables, $b$ denotes the length of a hash table, and $< |u|, |v| >$ denotes the bit lengths of the encrypted key-value pairs in bucket.*

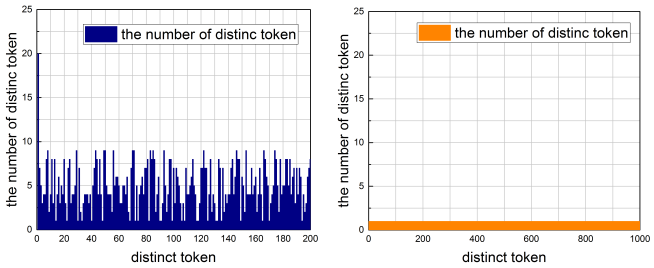
***Definition 6.2 (Leakage Function $\mathcal{L}_2$ for for Search Pattern).*** *Given a search object $q_i$, $\mathcal{L}_2 = (\pi(i,j), \mathcal{X})$, $\forall j \in [1, i-1]$ and $\mathcal{X}$ is the search distribution.*



(a) The Distribution of Encrypted uniform Dataset (b) The Distribution of Encrypted Non-uniform Dataset

Fig. 8: The Distribution of Encrypted Dataset

In this paper, we use a two-server setting to hide the query distribution effectively. As a result, all of the search tokens are distinct from the perspective of the server $B$, irrespective of whether these queries are the same. Consequently, server $B$ could not obtain valid information, even by statistical methods. As we defined in the Security Definitions section, the search pattern $\pi$ is defined by a binary matrix $\pi(i,j) = 1$ if the search token of query $q_i$ and the search token of query $q_j$ are the same and $\pi(i,j) = 0$ otherwise. Hence, search pattern $\pi$ is always $\pi(i,j) = 0$ if $i \neq j$. Fig. 9 shows pictorially the search distribution of SEND and previous schemes. We randomly perform 1,000 query requests, some of which are repetitive. We consider the same query request as $Q$. In this experiment, the 1000 query requests contain 200 $Q$s. Fig. 9(a) shows the search distribution $\mathcal{X}$ from the perspective of the server in the previous model, and Fig. 9(b) shows our perspective. From this, it is apparent that SEND can flatten the query distribution.




(a) Previous schemes (b) SEND

Fig. 9: Search distribution $\mathcal{X}$.

Cash et al. in [15] design a counting attack that an adversary can recover the query keyword if he or she knows the number of matched documents for each keyword in a targeted document set. In the scenario of retrieving high-dimensional data based on hash values, hash values can be regarded as keywords, and high-dimensional data sets can be regarded as document sets. Then a adversary can link the LSH values to the search tokens by comparing the distribution of observed search tokens $\mathcal{X}'$ to the known distribution of tokens $\mathcal{X}$. In this paper, our scheme can resist such attack due to the flattened search distribution.

***Definition 6.3 (Leakage Function $\mathcal{L}_3$ for Access Pattern).*** *Given a search object $q_i$, $\mathcal{L}_3 = (E(q_i), |resultDataID(q_i)|)$, $E(q_i)$ denotes the identifiers*

*of buckets being accessed and $resultDataID(q_i)$ denotes the size of the result.*

Specifically, the access pattern includes search results. In SEND, we put $bucketID$ in the bucket, encrypt it, and permute the index. We emphasize that the encrypted $DataIDList$ do not reveal additional information since $DataIDList$ are encrypted before the user recovery them. In the search phase, we search the index to find the matching $bucketIDs$ and get the corresponding $DataIDList$. The search process increases the retrieval time, but, considering that it improves the security, a small increase in the retrieval time is worthwhile.

***Theorem 1.*** *If $(Gen, Enc, Dec)$ is CPA-secure and $F$, $P$ are pseudo-random, then $\Omega$ with leakage functions $(\mathcal{L}_1, \mathcal{L}_2, \mathcal{L}_3)$ is secure against adaptive chosen-query attacks (CQA2) .*

*Proof:* We use the following simulations to proof the Theorem 1. □

**The simulation of the index construction:** Algorithm 3 describes the procedure of the simulation of the index construction. The simulator $\mathcal{S}$ generates a randomized index $\widetilde{\mathcal{I}}$ based on $\mathcal{L}_1$, which is the same size as the real encrypted index, $\mathcal{I}$. $\widetilde{\mathcal{I}}$ consists of $L \times |R|$ hash tables, and there are $b$ buckets in each hash table. Each simulated bucket consists of a random string with the same length as the real encrypted bucket. The random string $\widetilde{key}$ and $\widetilde{value}$ are generated by PRF. By reduction, differentiating $\mathcal{I}$ from $\widetilde{\mathcal{I}}$ implies distinguishing between PRF and a truly random function. Therefore, the probability of differentiating $\mathcal{I}$ from $\widetilde{\mathcal{I}}$ for adversary $\mathcal{A}$ is bounded by $Adv_{\mathcal{A}}^{prf}(\lambda)$. More formally,

$$Pr[Real_{\mathcal{A}}^{index}(\lambda) = 1] - Pr[Ideal_{\mathcal{A},\mathcal{S}}^{index}(\lambda) = 1] \leq Adv_{\mathcal{A}}^{prf}(\lambda) \tag{10}$$

**Algorithm 3** Simulation of the Index Construction

**Input:** $\mathcal{L}_1$
**Output:** $\widetilde{\mathcal{I}}$
**for** $i$ in $[1, L * |R|]$ **do**
  **for** $j$ in $[1, b]$ **do**
    $\widetilde{key} \xleftarrow{\$} \{0,1\}^{|u|}$
    $\widetilde{value} \xleftarrow{\$} \{0,1\}^{|v|}$
    $< \widetilde{key}, \widetilde{value} > \rightarrow \widetilde{\mathcal{I}}$
  **end for**
**end for**
**return** $\widetilde{\mathcal{I}}$

**The simulation of the search token generation:**

Algorithm 4 describes the procedure of the simulation of the search token generation. The simulator $\mathcal{S}$ generates random strings as simulated tokens $\widetilde{T}_q$ when the query object $q$ is sent. $\widetilde{T}_q$ is not computationally indistinguishable from $T_q$ due to the semantic security of secure symmetric encryption and PRF. For the following query objects, even if some of them are the same as the query object, $q$, the simulator, $\mathcal{S}$, generates new simulated tokens for each of the objects because all of the search tokens are distinct from the view of adversary $\mathcal{A}$ indicated in $\mathcal{L}_2$. Thus, $\mathcal{A}$ cannot differentiate between the simulated search tokens generated by $\mathcal{S}$ and the real search tokens. Hence,

$$Pr[Real_{\mathcal{A}}^{token}(\lambda) = 1] - Pr[Ideal_{\mathcal{A},\mathcal{S}}^{token}(\lambda) = 1] \leq Adv_{\mathcal{A}}^{prf}(\lambda) \tag{11}$$

**Algorithm 4** Simulation of the Search Token Generation

**Input:** $\mathcal{L}_1, \mathcal{L}_2, q$
**Output:** $\widetilde{T}_q$
**for** $i$ in $[1, L * |R|]$ **do**
  $\widetilde{x^*} \xleftarrow{\$} \{0,1\}^*$
  $\widetilde{x^{**}} \xleftarrow{\$} \{0,1\}^*$
  $< \widetilde{x^*}, \widetilde{x^{**}} > \rightarrow \widetilde{T}_q$
**end for**
**return** $\widetilde{T}_q$

**The simulation of the search:** Algorithm 5 describes the procedure of the simulation of the search. The simulator $\mathcal{S}$ returns $\widetilde{resultDataID}$ based on $\mathcal{L}_1$, $\mathcal{L}_2$, $L_3$ and $\widetilde{T}_q$. It gets the $\widetilde{value}$ in the buckets that $E(q)$ contains. In Algorithm 3 we already know that value is generated by the PRF. Thus, $\mathcal{A}$ cannot differentiate between the simulated search results $\widetilde{resultDataID}$ generated by $\mathcal{S}$ and the real results $resultDataID$. Therefore,

$$Pr[Real_{\mathcal{A}}^{search}(\lambda) = 1] - Pr[Ideal_{\mathcal{A},\mathcal{S}}^{search}(\lambda) = 1] \leq Adv_{\mathcal{A}}^{prf}(\lambda) \tag{12}$$

**Algorithm 5** Simulation of the Search

**Input:** $\mathcal{L}_1, \mathcal{L}_2, L_3, \widetilde{T}_q$
**Output:** $\widetilde{resultDataID}$
**for** each $i$ in $E(q)$ **do**
  $\widetilde{resultDataID} \leftarrow \widetilde{value}$ in bucket $i$ of $\widetilde{\mathcal{I}}$
**end for**
**return** $\widetilde{resultDataID}$

**Conclusion:** By combining all the contributions from the above simulations, We can say that for all probabilistic polynomial time adversaries $\mathcal{A}$, there exists a probabilistic polynomial time simulator $S$, such that:

$$Pr[Real_{\mathcal{A}}^{SEND}(\lambda) = 1] - Pr[Ideal_{\mathcal{A},\mathcal{S}}^{SEND}(\lambda) = 1] \leq Adv_{\mathcal{A}}^{prf}(\lambda) \tag{13}$$

Thus, our proposed scheme is secure against CQA2.

To improve the security of SEND, We further take two other measures. First, we fill the empty buckets with dummy paddings so that the server cannot distinguish between the effectiveness of different buckets. In addition, the results that are returned to data user are composed of the data objects that has higher similarity and some dummy paddings. When the user receives the results, he or she easily can get the right number of data objects of interest after encryption. Second, we permute the index at the end of each query. Because the method of retrieval is based on searching the index, the permutation of the index at the end of each query will not have an impact on the correctness of the result of the query. Also, it ensures that server *B* cannot learn the access frequency of each bucket through such frequent permutation. Even if the query objects are the same, their corresponding buckets that are being accessed are different, and increased randomness improves security.

Through the above proof and analysis, we have demonstrated that SEND has very high security. It can be used to implement a secure similarity search over an encrypted, high-dimensional dataset. What is more exciting is that the technique used to improve security in SEND can be applied to any dataset, irrespective of whether the distribution of the data is uniform.

TABLE 2: Characteristics of the four datasets

| Dataset | Number of objects | Dimension | Size of the dataset | $UDD$(Uniformity of Data Distribution) |
|---|---|---|---|---|
| FC | $5.8 \times 10^5$ | 54 | 72.2 MB | 70.30 |
| SD | $1.0 \times 10^6$ | 60 | 266 MB | 52.16 |
| ES | $1.0 \times 10^4$ | 179 | 7.15 MB | 33.99 |
| FK | $3.0 \times 10^4$ | 200 | 22.1 MB | 29.04 |

From a higher perspective of security, SEND is not enough to protect privacy if the server $A$ builds a profile of the user by observing the access pattern of the buckets. The leakage based on the sequence of storage locations accessed by the client is a common problem for outsourced storage applications. Oblivious RAM (ORAM) [44], as an algorithm that allows clients to hide their access pattern to the remote storage by constantly shuffling and re-encrypting as they access the data, is usually used to prevent the above leakage. However, the common ORAM often suffers from large bandwith overhead and large client storage so our scheme does not cover this part in this paper. Constructions that improve searchable encryption from this perspective can be found in [45]–[48].

## 7 EXPERIMENTAL EVALUATION

In this section, we present the experimental evaluation of our proposed scheme considering space cost, building index time, search time, recall, and precision. We perform the experiments on a machine with an Intel Core i5-2520M 2.50 GHz processor, 8 GB DRAM and a 500-GB 5400 RPM SATA disk.

### 7.1 Evaluation Datasets

We use four datasets to evaluate our scheme, i.e., three real datasets and a simulated dataset (SD). TABLE 2 sumamrizes the characteristics of the four datasets.

Forest Covertype (FC): The data in Forest Covertype are derived from a U.S. Forest Service and U.S. Geological Survey. The data in some of the columns (not scaled) are in raw form, and the data in other columns for qualitative independent variables are in binary (0 or 1) form. There are 580 K data objects in Forest Covertype, and each data object has 54 features. The Forest Covertype data are distributed non-uniformly, i.e., the data are skewed.

Simulated Dataset (SD): We generate an artificial dataset, i.e., the simulated dataset, SD. The simulated dataset, SD, has 1 M data objects, and each data object has 60 dimensions.

Epileptic Seizure Recognition Data Set (ES): This dataset is a pre-processed and re-structured version of a very commonly used dataset featuring epileptic seizure detection. Attribute Characteristics are integer and real.

Flickr30K (FK): This dataset consists of 30K images from Flickr. We extracts the pixel features of these images. Then following common practice, We use PCA to pre-process the dataset and keep 200 dimensions.

### 7.2 Evaluation Metrics

Since the time required to build the index, search time, recall, and precision our our main evaluation metrics, a detailed explanation of these evaluation metrics is presented below. Time required to build the index: The time required to build the index time includes the time of the selecting the radii and the time required to build $\mathcal{I}_D$ and $\mathcal{I}_R$.

Search Time: The search time describes the time cost between the users sending a query request and receiving the resulting set from the server.

Recall: Recall is an important criterion for judging whether the query result of a similarity search scheme is accurate. It is result that has been retrieved accurately divided by the total amount of an ideal result. The ideal result contains the points that are actually closest to the query under a distance metric. In this paper, we use Euclidean distance to measure the points that are actually nearest the query point. For a search object, $q$, we denote the result set by $R(q)$ and denote the ideal result set of $q$ by $I(q)$. Then, the formal definition of recall is:

$$Recall = \frac{|R(q) \cap I(q)|}{|I(q)|}. \tag{14}$$

Precision: Precision is another important criterion for judging whether the results of a query of a similarity search scheme are accurate. Precision is the fraction of the total amount of all of the retrieved results that is retrieved accurately. Therefore, we also can consider precision as hit rate. Specially, precision is defined by:

$$Precision = \frac{|R(q) \cap I(q)|}{|R(q)|}. \tag{15}$$

From the definitions of recall and precision, it is apparent that the best value for both recall and precision is 1. In this paper, our SEND focuses on improving recall and precision.

### 7.3 Performance Evaluation

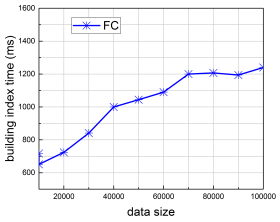
In this section, we first analysis computation complexity. After that, we discuss the experimental result. We randomly select 100 data point from each dataset as query points. Each evaluation metric is taken as the average value of all the query points evaluation results.

#### 7.3.1 Computation Complexity

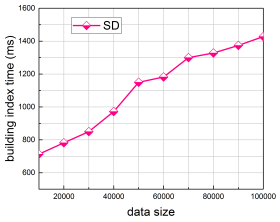
SEND involves 4 major steps: selecting the optimal radius, building index, constructing trapdoor and performing search request. Selecting the optimal radius is the first step and the computation complexity of it is $O(n^2)$. In the process of building index, we need to calculate $L \cdot M \cdot |R|$ hash functions for each data, so the computational complexity is $O(n \cdot L \cdot M \cdot |R|)$. Similarly, the computational complexity of constructing trapdoor is $O(L \cdot M \cdot |R|)$. Since the buckets of all hash tables are encrypted, we need to find results through the traversal method when performing the query operation. Thus the time complexity of performing search request is $O(L \cdot |R| \cdot b)$.

#### 7.3.2 Space Cost

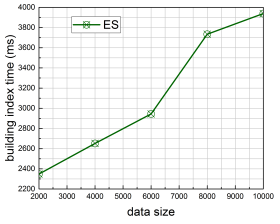
TABLE 3 lists the storage overhead of $\mathcal{I}_D$, $\mathcal{I}_R$ and trapdoor. From the table we can see that when the parameters $L$ and $|R|$ are the same, that is, the number of hash tables is the same, the more data the data set, the larger the index $\mathcal{I}_D$ space overhead, but the difference in the cost of the index $\mathcal{I}_R$ and the trapdoor $T_q$ is not very large. For the index $\mathcal{I}_D$, since the identifiers of the data object are to be stored therein, a large amount of data necessarily results in an index with larger space overhead. But for the index $\mathcal{I}_R$, it only needs to store the corresponding $bucketID$ values and the random numbers, so when the number of hash table and the length of a hash table are determined, the index $\mathcal{I}_R$ is basically determined. The overhead of the trapdoor $T_q$ is mainly related to the number of hash functions $L \times |R|$. How many hash functions mean how many hashes are included in the $T_q$.



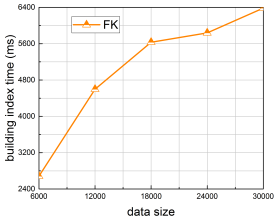

(a) FC




(b) SD




(c) ES




(d) FK

Fig. 10: Time required to build indexes for each dataset.

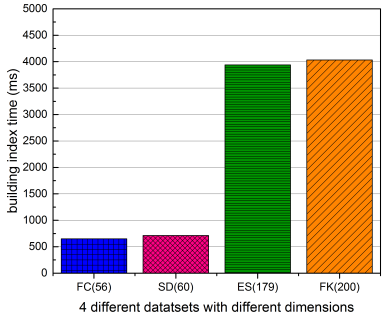


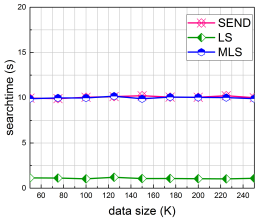

Fig. 11: Relationship between building index time and dimension(datasize=10000, $L = 55$, $|R| = 10$).

#### 7.3.3 Construction Time

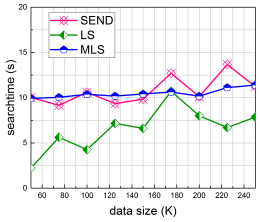
Fig. 10 shows the relationship between the time required to build the index and the size of the data set on the four datasets. The four results of (a) to (d) show that the time required to build the index is basically linear with the size of the data set. Thus, we can say that the time required to build the index increases linearly with the size of the dataset. In other words, the larger the data set, the more time it takes to build an index. Because the calculation time for a point is basically the same, so the more points you need, the more time it takes.

In addition, we also did an experiment on the impact of data dimensions on indexing time. The result is shown in fig. 11. The size of the four datasets and the size of the hash tables are fixed to be the same. We can see that the construction time of the 56-dimensional dataset SD and the 60-dimensional dataset SD is basically the same, and the construction time of the 179-dimensional dataset ES and the 200-dimensional dataset FK is not much different. In general, the larger the data dimension, the more time it takes to build an index. This is mainly because the data points with large dimensions take longer to do the inner product of Equation 1, which results in a longer time to construct index.

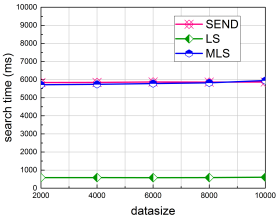
In order to ensure the maximum security of the data, we choose to encrypt the dataset locally, and then upload the encrypted dataset to the cloud server. We observed the time required to encrypt dataset locally, as shown in TABLE 4. From the table we can see that the time to encrypt the dataset locally is not very long, so encrypting the dataset locally is an option that is both efficient and secure.



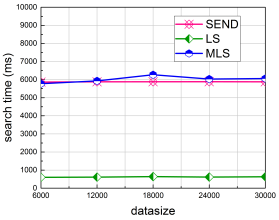

(a) FC




(b) SD




(c) ES




(d) FK

Fig. 12: Search time per query point.

#### 7.3.4 Search Time

Fig. 12 shows the search time on four datasets of our scheme, the LS (LSH-based SSE) scheme, and the MLS (multi-r-LSH-based SSE) scheme. The LS scheme uses a fixed radius and a hash table, and the MLS scheme uses a fixed radius and the same number of hash tables as our scheme. As illustrated, the trend of the performance of these three schemes on both datasets remains basically constant.

TABLE 3: Storage overhead

| Dataset | Datasize | Dimension | $L$ | $\|R\|$ | $\mathcal{I}_D$ stoarge | $\mathcal{I}_R$ stoarge | trapdoor stoarge |
|---|---|---|---|---|---|---|---|
| FC | $5 \times 10^5$ | 54 | 55 | 10 | 295MB | 5.05MB | 5.91KB |
| SD | $1.0 \times 10^5$ | 60 | 55 | 10 | 149MB | 5.34MB | 5.91KB |
| ES | $1.0 \times 10^4$ | 179 | 55 | 10 | 117MB | 3.08MB | 5.90KB |
| FK | $3.0 \times 10^4$ | 200 | 55 | 10 | 125MB | 3.95MB | 5.91KB |

TABLE 4: Time required for encryption of different datasets

| Dataset | Datasize | Dimension | Encryption time |
|---|---|---|---|
| FC | $5 \times 10^5$ | 54 | 465.239s |
| SD | $1.0 \times 10^5$ | 60 | 96.474s |
| ES | $1.0 \times 10^4$ | 179 | 26.566s |
| FK | $3.0 \times 10^4$ | 200 | 99.066s |

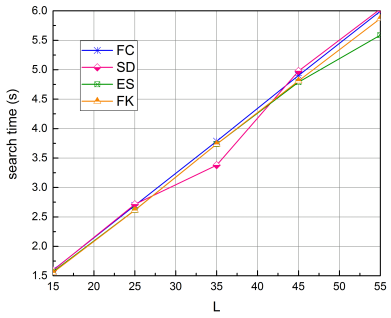



Fig. 13: Relationship between search time and $L$(FC's datasize=500000, SD's datasize=100000, ES's datasize=10000, FK's datazie=30000, $|R| = 10$ for all dataset.

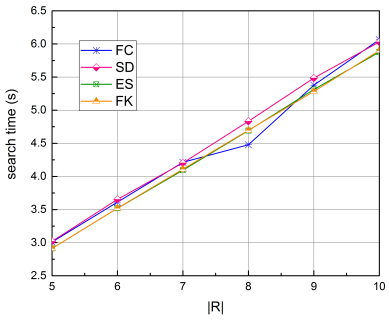



Fig. 14: Relationship between search time and $|R|$(FC's datasize=500000, SD's datasize=100000, ES's datasize=10000, FK's datazie=30000, $L = 55$ for all dataset.

This is because all three scheme are based on LSH, and all data objects in the dataset are indexed by hash tables. Therefore, no matter how many data points present in the dataset, the search time of a data point is only related to the hash table; it has nothing to do with the number of data points. Note that the search time of our solution and the search time of MLS are comparable because they both has hash tables with multiple different $r$. And LS only has hash tables with a single fixed $r$, so its search time is less than SEND and MLS.

Since the search time is not related to the number of points in the dataset, we naturally need to explore which parameters it is related to. As shown in Fig. 13, the search time increases linearly with the number of concatenation functions $L$, and it also increases linearly with the length of radius set $|R|$ as shown in fig. 14. In addition, The growth curves of these four data sets are very close, thus further indicating that when the number and length of hash tables are the same, the search time can be basically determined.

### 7.3.5 Recall and Precision

In the above experiment concerning search time, SEND did not lag behind the other schemes. Below, we present the salient advantages of SEND and its search quality in the experiment. Two important metrics for evaluating search quality are recall and precision.

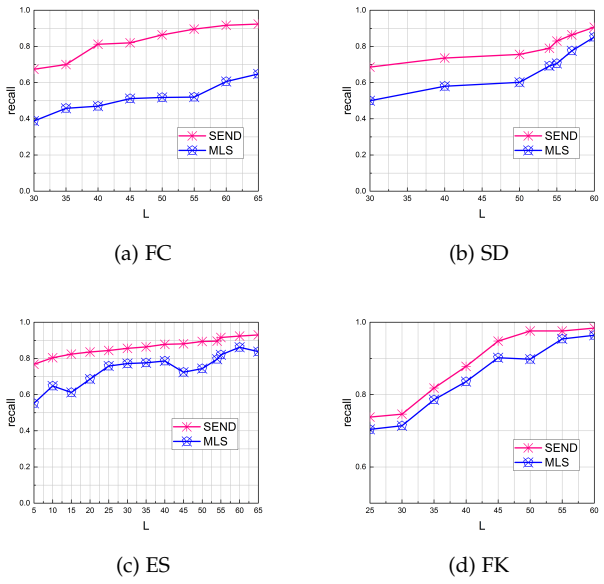
Fig. 15 shows the change in recall for SEND and MLS over four datasets as $L$ changes. It is apparent that increases in $L$ make recall greater, i.e., the quality of the search is improved. The larger the $L$, i.e., the greater the number of hash tables, means that the more buckets can be probed at the time of the query, which increases the probability of the exact KNN (K-Nearest Neighbor) of the query object being found. Thus the recall rate naturally increases. In comparing SEND and the scheme based on general LSH, Fig. 15 shows that SEND has better search quality. The recall of SEND always is higher than the recall of the scheme based on general LSH. In addition, SEND can use a smaller $L$ to achieve a recall that MLS requires a larger $L$. That is to say, when setting an expected recall rate, using SEND can save space cost.

For most schemes, the process of selecting approximate nearest neighbors is to select some candidates by approximate method, and then select KNN from the candidates using Euclidean distance. In this process, the number of candidates is usually allowed to be larger than K. For example, by selecting 10 points from a data set with 1M elements, we can select 1K points first, and then 10 points from 1K points. In this way, both efficiency and accuracy can be guaranteed. Therefore, the recall rate can roughly determine the quality of a search scheme. However, we still did an experiment for precision on FC.

Fig. 16 shows the change in recall for SEND and MLS

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/TCC.2020.3000233, IEEE Transactions on Cloud Computing

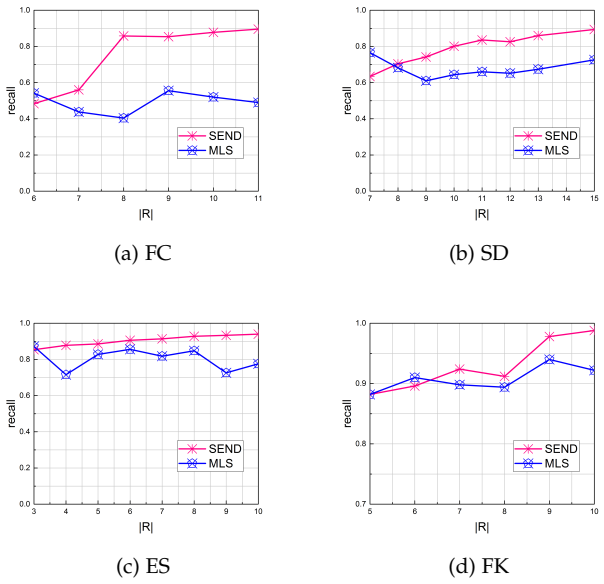





(a) FC (b) SD

(c) ES (d) FK

Fig. 15: Comparison of SEND and MLS with recall and relationship between recall and $L$.

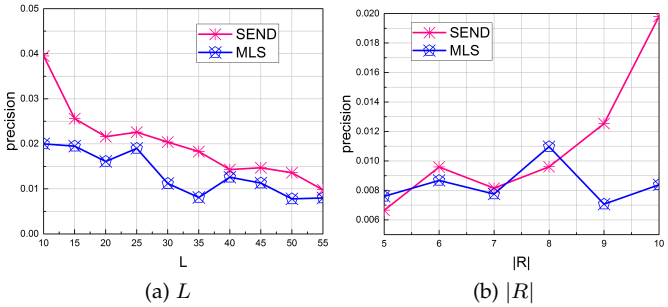



(a) FC (b) SD

(c) ES (d) FK

Fig. 16: Comparison of SEND and MLS with recall and relationship between recall and $|R|$.

over four datasets as $|R|$ changes. The results further show that the recall of SEND is better than MLS. $|R|$ in our scheme represents the number of different radii, and $|R|$ in MLS represents the number of the same radii. Since in our scheme, the radius of each point is determined according to the surrounding density, the more the radius indicates that the radius selection of each point can be more accurate, so there is a greater probability to be neighbors for the points in one bucket. Therefore as the number of optional radii $R$ increases, the recall for SEND increases. For MLS, although the number of elements in the radius set $R$ increases, the




(a) $L$ (b) $|R|$

Fig. 17: Precision(dataset: FC).

elements in the $R$ set are the same value, so the recall does not change significantly with the change of $|R|$.

Fig. 17(a) shows that, as $L$ increases, the precision of SEND and of the scheme based on general LSH decreases. This is because the larger $L$ becomes, the more hash tables there will be in the index. And the multiple of the total number of returned results will be larger than the multiple of the number of results that are retrieved accurately. However, the precision of SEND is always higher than the precision of the scheme based on general LSH. Fig. 17(b) shows that, as $|R|$ increases, the precision of SEND increases, but the precision of the scheme based on general LSH has no noticeable increasing or decreasing trend. In SEND, a suitable radius is chosen for each point, and the larger $|R|$ becomes, the more accurate the radius of each point becomes, so the precision will be improved. Instead of considering the best radius for each point, the scheme based on general LSH uses a uniform radius for each point. Therefore, the change of $R$ has no definite effect on its precision. When $R$ is greater than a certain value, the precision of SEND is obviously greater than the precision of the scheme based on general LSH, and it increases rapidly.

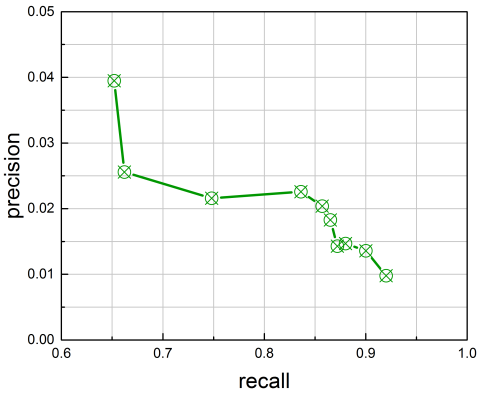



Fig. 18: Relationship between recall and precision.

During the experiment, we also observe the law shown in Fig. 18. If the number of returned points to get a good recall is low, then the precision of the algorithm is high. It is a corollary of the previous discussion.

The experiment described above shows that our scheme is comparable to common solutions in terms of search time and superior to other schemes in terms of both recall and precision, so our scheme can be used extensively.

## 8 Conclusion

We investigated the problem of similarity search over an encrypted, non-uniform datasets. Then, we proposed a novel scheme to improve the search quality and security for non-uniform datasets. Our scheme proved to be effective against adaptive, chosen- query attacks, and it has higher security than other schemes for any dataset. We implemented our design and evaluate its search time, recall, and precision. The results showed that our scheme provides very high quality searches of encrypted, non-uniform datasets.

## Acknowledgment

This paper is supported by the National Natural Science Foundation of China under grant No. 61501080, 61572095, 61871064 and 61702105 and U1804263.

## References

[1] D. X. Song, D. A. Wagner, and A. Perrig, "Practical techniques for searches on encrypted data," in *2000 IEEE Symposium on Security and Privacy, Berkeley, California, USA, May 14-17, 2000*, 2000, pp. 44–55.

[2] Z. Fu, F. Huang, K. Ren, J. Weng, and C. Wang, "Privacy-preserving smart semantic search based on conceptual graphs over encrypted outsourced data," *IEEE Trans. Information Forensics and Security*, vol. 12, no. 8, pp. 1874–1884, 2017.

[3] Z. Fu, X. Wu, C. Guan, X. Sun, and K. Ren, "Toward efficient multi-keyword fuzzy search over encrypted outsourced data with accuracy improvement," *IEEE Trans. Information Forensics and Security*, vol. 11, no. 12, pp. 2706–2716, 2016.

[4] V. B. Lima and S. R. Maniyath, "Geometric location finder based on encrypted spatial data using geometric range queries," in *2018 International Conference on Design Innovations for 3Cs Compute Communicate Control (ICDI3C)*. IEEE, 2018, pp. 75–79.

[5] H. Cui, X. Yuan, and C. Wang, "Harnessing encrypted data in cloud for secure and efficient mobile image sharing," *IEEE Trans. Mob. Comput.*, vol. 16, no. 5, pp. 1315–1329, 2017.

[6] J. Vaidya and J. Li, Eds., *Algorithms and Architectures for Parallel Processing - 18th International Conference, ICA3PP 2018, Guangzhou, China, November 15-17, 2018, Proceedings, Part IV*, ser. Lecture Notes in Computer Science, vol. 11337. Springer, 2018.

[7] M. Kuzu, M. S. Islam, and M. Kantarcioglu, "Efficient similarity search over encrypted data," in *IEEE 28th International Conference on Data Engineering (ICDE 2012), Washington, DC, USA (Arlington, Virginia), 1-5 April, 2012*, 2012, pp. 1156–1167.

[8] Z. Fu, J. Shu, J. Wang, Y. Liu, and S. Lee, "Privacy-preserving smart similarity search based on simhash over encrypted data in cloud computing," *ŁŁ*, vol. 16, no. 3, pp. 453–460, 2015.

[9] G. Pernul, P. Y. A. Ryan, and E. R. Weippl, Eds., *Computer Security - ESORICS 2015 - 20th European Symposium on Research in Computer Security, Vienna, Austria, September 21-25, 2015, Proceedings, Part II*, ser. Lecture Notes in Computer Science, vol. 9327. Springer, 2015.

[10] V. Vianu, Ed., *Proceedings of the Thirteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, May 24-26, 1994, Minneapolis, Minnesota, USA*. ACM Press, 1994.

[11] *2008 IEEE Symposium on Security and Privacy (S&P 2008), 18-21 May 2008, Oakland, California, USA*. IEEE Computer Society, 2008.

[12] H. He, W. Zhang, and S. Zhang, "A novel ensemble method for credit scoring: Adaption of different imbalance ratios," *Expert Syst. Appl.*, vol. 98, pp. 105–117, 2018.

[13] P. K. Chan and S. J. Stolfo, "Toward scalable learning with non-uniform class and cost distributions: A case study in credit card fraud detection." in *KDD*, vol. 98, 1998, pp. 164–168.

[14] C. J. Martinez, *Hash algorithm design for a non-uniformly distributed database*. The University of Texas at San Antonio, 2008.

[15] D. Cash, P. Grubbs, J. Perry, and T. Ristenpart, "Leakage-abuse attacks against searchable encryption," *IACR Cryptology ePrint Archive*, vol. 2016, p. 718, 2016.

[16] Y. Chang and M. Mitzenmacher, "Privacy preserving keyword searches on remote encrypted data," in *Applied Cryptography and Network Security, Third International Conference, ACNS 2005, New York, NY, USA, June 7-10, 2005, Proceedings*, 2005, pp. 442–455.

[17] Q. Wang, M. He, M. Du, S. S. M. Chow, R. W. F. Lai, and Q. Zou, "Searchable encryption over feature-rich data," *IEEE Trans. Dependable Sec. Comput.*, vol. 15, no. 3, pp. 496–510, 2018.

[18] E. Goh, "Secure indexes," *IACR Cryptology ePrint Archive*, vol. 2003, p. 216, 2003.

[19] R. Curtmola, J. A. 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, CCS 2006, Alexandria, VA, USA, Iooctober 30 - November 3, 2006*, 2006, pp. 79–88.

[20] J. Li, Q. Wang, C. Wang, N. Cao, K. Ren, and W. Lou, "Enabling efficient fuzzy keyword search over encrypted data in cloud computing," *IACR Cryptology ePrint Archive*, vol. 2009, p. 593, 2009.

[21] H. Park, B. H. Kim, D. H. Lee, Y. D. Chung, and J. Zhan, "Secure similarity search," in *2007 IEEE International Conference on Granular Computing, GrC 2007, San Jose, California, USA, 2-4 November 2007*, 2007, p. 598.

[22] C. Wang, K. Ren, S. Yu, and K. M. R. Urs, "Achieving usable and privacy-assured similarity search over outsourced cloud data," in *Proceedings of the IEEE INFOCOM 2012, Orlando, FL, USA, March 25-30, 2012*, 2012, pp. 451–459.

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

[24] W. Sun, B. Wang, N. Cao, M. Li, W. Lou, Y. T. Hou, and H. Li, "Verifiable privacy-preserving multi-keyword text search in the cloud supporting similarity-based ranking," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 11, pp. 3025–3035, 2014.

[25] D. Cash, S. Jarecki, C. S. Jutla, H. Krawczyk, M. Rosu, and M. Steiner, "Highly-scalable searchable symmetric encryption with support for boolean queries," in *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part I*, 2013, pp. 353–373.

[26] A. Gionis, P. Indyk, and R. Motwani, "Similarity search in high dimensions via hashing," in *VLDB'99, Proceedings of 25th International Conference on Very Large Data Bases, September 7-10, 1999, Edinburgh, Scotland, UK*, 1999, pp. 518–529.

[27] J. Qian, Z. Huang, Q. Zhu, and H. Chen, "Hamming metric multi-granularity locality-sensitive bloom filter," *IEEE/ACM Transactions on Networking*, vol. 26, no. 4, pp. 1660–1673, 2018.

[28] W. W. Ng, S. Lei, and X. Tian, "Two-layer localized sensitive hashing with adaptive re-ranking," in *2018 International Conference on Wavelet Analysis and Pattern Recognition (ICWAPR)*. IEEE, 2018, pp. 223–230.

[29] Y. Ma, X. Feng, Y. Liu, and S. Li, "Bch–lsh: a new scheme of locality-sensitive hashing," *IET Image Processing*, vol. 12, no. 6, pp. 850–855, 2018.

[30] X. Yuan, X. Wang, C. Wang, C. Yu, and S. Nutanong, "Privacy-preserving similarity joins over encrypted data," *IEEE Trans. Information Forensics and Security*, vol. 12, no. 11, pp. 2763–2775, 2017.

[31] M. Naveed, M. Prabhakaran, and C. A. Gunter, "Dynamic searchable encryption via blind storage," in *2014 IEEE Symposium on Security and Privacy, SP 2014, Berkeley, CA, USA, May 18-21, 2014*, 2014, pp. 639–654.

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

[33] E. Stefanov, C. Papamanthou, and E. Shi, "Practical dynamic searchable encryption with small leakage," in *21st Annual Network and Distributed System Security Symposium, NDSS 2014, San Diego, California, USA, February 23-26, 2014*, 2014.

[34] Q. Wang, S. R. Kulkarni, and S. Verdú, "Divergence estimation for multidimensional densities via k-nearest-neighbor distances," *IEEE Trans. Information Theory*, vol. 55, no. 5, pp. 2392–2405, 2009.

[35] J. Gao, H. V. Jagadish, W. Lu, and B. C. Ooi, "DSH: data sensitive hashing for high-dimensional k-nnsearch," in *International Conference on Management of Data, SIGMOD 2014, Snowbird, UT, USA, June 22-27, 2014*, 2014, pp. 1127–1138.

[36] J. Gao, H. V. Jagadish, B. C. Ooi, and S. Wang, "Selective hashing: Closing the gap between radius search and k-nn search," in *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Sydney, NSW, Australia, August 10-13, 2015*, 2015, pp. 349–358.

[37] J. Katz and Y. Lindell, "Introduction to modern cryptography: principles and protocols. cryptography and network security," 2008.

[38] K. S. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft, "When is "nearest neighbor" meaningful?" in *Database Theory - ICDT '99, 7th International Conference, Jerusalem, Israel, January 10-12, 1999, Proceedings.*, 1999, pp. 217–235.

[39] Y. Elmehdwi, B. K. Samanthula, and W. Jiang, "Secure k-nearest neighbor query over encrypted data in outsourced environments," in *IEEE 30th International Conference on Data Engineering, Chicago, ICDE 2014, IL, USA, March 31 - April 4, 2014*, 2014, pp. 664–675.

[40] A. Liu, K. Zheng, L. Li, G. Liu, L. Zhao, and X. Zhou, "Efficient secure similarity computation on encrypted trajectory data," in *31st IEEE International Conference on Data Engineering, ICDE 2015, Seoul, South Korea, April 13-17, 2015*, 2015, pp. 66–77.

[41] H. Cui, X. Yuan, Y. Zheng, and C. Wang, "Enabling secure and effective near-duplicate detection over encrypted in-network storage," in *35th Annual IEEE International Conference on Computer Communications, INFOCOM 2016, San Francisco, CA, USA, April 10-14, 2016*, 2016, pp. 1–9.

[42] S. Kamara, C. Papamanthou, and T. Roeder, "Dynamic searchable symmetric encryption," in *the ACM Conference on Computer and Communications Security, CCS'12, Raleigh, NC, USA, October 16-18, 2012*, 2012, pp. 965–976.

[43] M. Chase and S. Kamara, "Structured encryption and controlled disclosure," *IACR Cryptology ePrint Archive*, vol. 2011, p. 10, 2011.

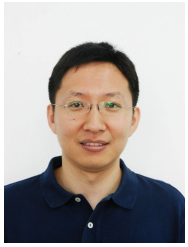
[44] O. Goldreich and R. Ostrovsky, "Software protection and simulation on oblivious rams," *J. ACM*, vol. 43, no. 3, pp. 431–473, 1996.

[45] M. T. Goodrich, M. Mitzenmacher, O. Ohrimenko, and R. Tamassia, "Privacy-preserving group data access via stateless oblivious RAM simulation," in *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, Kyoto, Japan, January 17-19, 2012*, 2012, pp. 157–167.

[46] E. Stefanov, M. van Dijk, E. Shi, T. H. Chan, C. W. Fletcher, L. Ren, X. Yu, and S. Devadas, "Path ORAM: an extremely simple oblivious RAM protocol," *J. ACM*, vol. 65, no. 4, pp. 18:1–18:26, 2018.

[47] A. Shafiee, R. Balasubramonian, M. Tiwari, and F. Li, "Secure DIMM: moving ORAM primitives closer to memory," in *IEEE International Symposium on High Performance Computer Architecture, HPCA 2018, Vienna, Austria, February 24-28, 2018*, 2018, pp. 428–440.

[48] R. Wang, Y. Zhang, and J. Yang, "D-ORAM: path-oram delegation for low execution interference on cloud servers with untrusted memory," in *IEEE International Symposium on High Performance Computer Architecture, HPCA 2018, Vienna, Austria, February 24-28, 2018*, 2018, pp. 416–427.



**Cheng Guo** (M'19) received the B.S. degree in computer science from Xian University of Architecture and Technology in 2002. He received the M.S. degree in 2006 and his Ph.D in computer application and technology, in 2009, both from the Dalian University of Technology, Dalian, China. From July 2010 to July 2012, he was a post doc in the Department of Computer Science at the National Tsing Hua University, Hsinchu, Taiwan. Since 2013, he has been an associate professor in the School of Software Technology at the Dalian University of Technology. His current research interests include information security, cryptology and cloud security.

**Wanping Liu** received the B.S. degree in software engineering from Dalian University of Technology, Dalian, China, in 2017. She is working towards the Master degree in the School of Software Technology at the Dalian University of Technology. Her research interests include cloud security, information security and applied cryptography.

**Ximeng Liu** (S13-M16) received the B.Sc. degree in electronic engineering from Xidian University, Xian, China, in 2010 and the Ph.D. degree in Cryptography from Xidian University, China, in 2015. Now he is the full professor in the College of Mathematics and Computer Science, Fuzhou University. Also, he is a research fellow at the School of Information System, Singapore Management University, Singapore. He has published more than 100 papers on the topics of cloud security and big data security‡including papers in IEEE Transactions on Computers, IEEE Transactions on Industrial Informatics, IEEE Transactions on Dependable and Secure Computing, IEEE Transactions on Service Computing, IEEE Internet of Things Journal, and so on. He awards Minjiang Scholars Distinguished Professor, Qishan Scholars in Fuzhou University, and ACM SIGSAC China Rising Star Award (2018). His research interests include cloud security, applied cryptography and big data security. He is a member of the IEEE, ACM, CCF.

**Yinghui Zhang** (M'18) received the Ph.D. degree in cryptography from Xidian University, China, in 2013. He is currently an Associate Professor with the National Engineering Laboratory for Wireless Security, Xian University of Posts and Telecommunications. He is also a Research Fellow with Singapore Management University. He has published over 50 research articles including ASIACCS, ACISP, Computer Networks, the IEEE INTERNET OF THINGS JOURNAL, and Computers & Security. His research interests include cloud security, public key cryptography, and wireless network security.