Fast and Secure kNN Query Processing in Cloud Computing

Xinyu Lei*, Guan-Hua Tu*, Alex X. Liu†, Tian Xie*
* Dept. of Computer Science and Engineering, Michigan State University, USA
† Ant Financial Service Group, Hangzhou, China
Email: {leixinyu, xietian1, ghtu}@msu.edu, xiangyangalex.lxy@antfin.com

***Abstract*—Advances in sensing and tracking technology lead to the proliferation of location-based services. Location service providers (LSPs) often resort to commercial public clouds to store the tremendous geospatial data and process location-based queries from data users. To protect the privacy of LSP's geospatial data and data user's query location against the untrusted cloud, they are required to be encrypted before sending to the cloud. Nevertheless, it is not easy to design a fast and secure location-based query processing scheme over the encrypted data. In this paper, we propose a Fast and Secure kNN (FSkNN) scheme to support secure k nearest neighbor (kNN) search in cloud computing. We reveal the inherent connection between an SkNN protocol and a secure range query protocol and further describe how to construct FSkNN based on a secure range query protocol. FSkNN leverages a customized accuracy-assured strategy to ensure the result accuracy and adopts a data structure named random Bloom filter (RBF) to build a secure index for efficiently searching. We formally prove the security of FSkNN under the random oracle model. Our evaluation results show that FSkNN is highly practical.**

I. Introduction

In a location-based service, the local service provider (LSP) hosts a big geospatial database that contains the information for massive locations. A user can send her current location to the LSP and then the LSP returns her the query results (e.g., the top 5 nearest hotels). One common practice for LSPs is to outsource the geospatial database to a powerful public cloud for both geospatial database storage and location-based queries processing. Cloud computing is identified as the next-generation computing paradigm, which can bring LSPs many benefits like lower operation fees and better performance. Nevertheless, LSPs would lose direct control over their data if they outsource their geospatial data to the public cloud, which leads to security issues that hinder the popularization of the new computing paradigm. For instance, the cloud may collect the location of the data user (i.e., the querier) and track the data user. Moreover, the cloud may be hacked and the stored data may be leaked. The leaked data allows the adversaries for commercial benefits or criminal purposes or gain improper benefits. To ensure LSP's data privacy, the geospatial data is required to be encrypted before sending to the cloud. Such a requirement is enforced by emerging laws (e.g., General Data Protection Regulation 2018 [1], [2]). However, the enforcement of data encryption reduces the data utility because it is usually less efficient to process location-based queries over the encrypted data. In this paper, we focus on location-based secure k nearest neighbor (SkNN) query, which is a ubiquitous query in real-world applications (e.g., search for the top 5 nearest hotels). Therefore, it is imperative to devise techniques to provide strong security against the untrusted cloud while still maintaining the cloud's kNN search performance over the encrypted database.

The studied problem is gradually formulated as follows. **Geospatial Database.** The LSP (i.e., data owner) hosts a geospatial database that contains n data items. Each geospatial data item is composed of two types of attributes. The first type is the spatial attribute (location information) and the second type is the non-spatial attribute(s) (non-location information). For example, the spatial attribute can be a hotel location and the non-spatial attributes may include the hotel name, price information, and its customer reviews. Each data item can be indexed and represented by its spatial information, which is a point in the two-dimensional (2-D) geospatial space. Hence, we can also use the term "point" to represent a data item in this paper. Let $S = \{p_1, \cdots, p_n\}$ represent the set of all points. A point p_i can also be treated as a vector from the origin point O to p_i, represented as $\overrightarrow{p_i}$. **SkNN Search.** There are three types of entities: a data owner, a cloud, and a group of data users. The data owner sends the database to the public cloud for storage. On receiving a query point q from a data user, the cloud processes kNN query and returns the k nearest points to the data user. The Euclidean distance is used as the distance metric. The SkNN queries processing problem is: the cloud should assist the data owner to store geospatial data and answer users' kNN queries while the cloud cannot learn useful information about the data owner's geospatial data and the data users' query point locations. **Threat Model.** We assume that the cloud is the adversary who is semi-honest (a.k.a., honest-but-curious) [3], [4]. In the semi-honest model, the cloud does not deviate from the defined protocol. However, the cloud tries to use legitimately received data to infer other private or sensitive information. For instance, the cloud may want to infer the data owner's geospatial data and the data users' query locations. Consequently, to protect the data owner's geospatial data privacy and data user's query location privacy, both the geospatial data and query location should be encrypted before sending to the public cloud.

More specifically, the proposed FSkNN service model is shown in Fig. 1. In FSkNN scheme, the data users can obtain

978-1-7281-4760-4/20/$31.00 ©2020 IEEE

the right to use the SkNN query service by requesting the secret keys from the data owner. To support fast and secure query processing, the data owner builds a secure index using the spatial attributes of all data items. Then, the data owner encrypts the whole data item. The pointers are used to record the association between the secure index item and the encrypted data item. Next, the data owner sends the secure index and the encrypted data items to the public cloud, which is responsible for data storage and query processing. Subsequently, the data users can generate valid search tokens and send them to the cloud for SkNN search. On receiving them, the cloud performs the search and sends back the results. Finally, the data user performs some post-processing operations (e.g., data item decryption) and gets the desirable SkNN results.

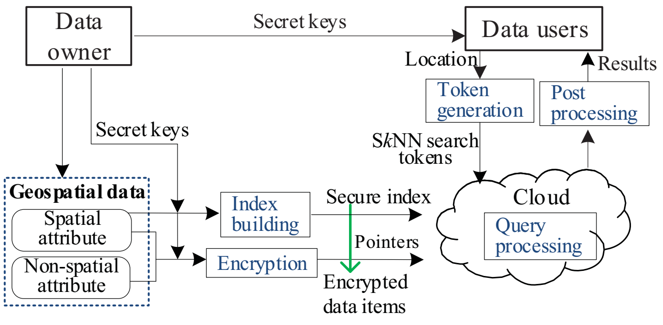


Fig. 1. FSkNN service model.

Two design goals are figured out for FSkNN.

- *Security.* FSkNN should preserve the following three types of privacy. (1) Data privacy: from the encrypted data items, the adversary cannot reveal any useful information about the data. (2) Index privacy: from the secure index, the adversary cannot learn any useful information about the data items. (3) Token privacy: from the encrypted search token, the adversary cannot infer any useful information about the data user's location information.
- *Efficiency.* FSkNN should satisfy two types of efficiency requirements. (1) Low query processing time: the data user can get the query results within a reasonable amount of time. (2) Low interaction rounds: the number of interaction rounds between the data user and the cloud should be small. It is desirable that the protocol only requires a small constant number of interaction rounds.

The major challenge for FSkNN is that it is difficult to achieve strong security and query processing efficiency simultaneously. The enforcement of data encryption on the cloud reduces the data utility. Thus, it becomes thorny to efficiently processing kNN queries on the encrypted data. A naive solution is that the data user encrypts the query point q and then sends to the cloud. The cloud computes the Euclidean distances between the encrypted query point q and the encrypted points in the database and then sorts the computed distances to get the query results. However, this solution is impossible to achieve strong security and query processing efficiency simultaneously. If a strong encryption algorithm, e.g., FHE (fully homomorphic encryption) [5], is employed, computing distance between strongly encrypted points is inefficient and hence search performance is low. If a weak encryption algorithm, such as OPE (order-preserving encryption) [6], is applied, then the strong security goal is breached. To tackle this conflict, we adopt the index-aid approach to sidestep the complex computations over the encrypted data. In this approach, the geospatial data items can be formally encrypted by standard strong encryption methods (e.g., AES) to achieve strong data privacy while search operations can be performed over a secure index. One issue of this approach is that it is still time-consuming to perform complex operations (i.e., computing Euclidean distance) over a strongly encrypted index. To avoid complex computations over the secure index, our key idea is to adopt the simpler range queries to realize the complex kNN search functionalities. As a result, FSkNN can achieve strong security and fast query processing simultaneously. The next problem is how to design a kNN protocol from a range query protocol. FSkNN addressed this problem by adopting the prefix membership verification technique [18] with proper customized design. Moreover, FSkNN employs random Bloom filter (RBF)-based data structure to build a secure index. To support efficient query processing, FSkNN organizes the RBFs into a binary tree structure so that the query processing can be finished in sublinear time.

TABLE I
COMPARISON BETWEEN FOUR PREVIOUS SCHEMES AND FSKNN
(●: SUPPORTED, ○: NOT SUPPORTED).

	[7]	[8]	[9]	[10]	FSkNN
Strong security	●	○	●	●	●
Sublinear time	○	●	●	●	●
Accurate results	●	●	●	○	●
Support $k > 1$	●	●	○	●	●
High dimensions	●	●	○	○	●
No local index	●	●	○	●	●
Single server	○	●	●	●	●
Interaction rounds	$O(1)$	$O(1)$	$O(1)$	$O(1)$	$O(1)$

We compare FSkNN with four previous SkNN schemes [7]–[10]. Table I summarizes the comparison results, which show that FSkNN outperforms the previous art. The limitations of previous solutions are explained as follows. Elmehdwi et al. [7] proposed a secure kNN protocol using a twin-cloud model [11] and Paillier cryptosystem [12]. This protocol employs too many expensive cryptographic operations, so its query processing time is too long to be used in reality (especially for large datasets). Wang et al. [8] developed a secure kNN scheme based on order-preserving encryption (OPE). The OPE only has weak privacy guarantees and therefore it fails to resist many real-world attacks [13]. Yao et al. [9] designed a solution that can support secure 1NN search by exploiting the Voronoi diagram [14] for space division. This scheme requires each data user to store a local index for searching, but the large-size local index is not desirable for weak users. Recently, a projection function-based approach is proposed in [10] to address SkNN problem. However, this approach suffers from two major limitations. First, it cannot ensure the results accuracy; only approximated results are supported. Second, it cannot handle high-dimensional data.

Three major contributions of this paper are identified.

- We develop a novel mechanism which converts a secure kNN problem into a secure range query problem. Accordingly, we propose FSkNN scheme, which preserves both query efficiency and security for kNN search.
- We design an accuracy-assured strategy on top of FSkNN scheme. The strategy enables FSkNN to provide users with accurate results while high query efficiency and security are still supported.
- We figure out the information leakage in FSkNN and formally prove that FSkNN scheme achieves security in the random oracle model based on the simulation proof technique.

II. kNN Protocol in Plaintext Domain

In this section, we first introduce how to construct a kNN query processing protocol from a range query processing protocol. Then, we develop a strategy to ensure the results accuracy of the constructed kNN protocol. Last, the parameter choices of the constructed kNN protocol are discussed. Note that this section does not consider the data privacy protection. We assume that the cloud hosts the geospatial database in plaintext and the data user submits range queries in plaintext to the cloud for a kNN search.

A. kNN Query from Range Query

1) 1-D Data: We show that a kNN query processing protocol in 1-D data can be constructed by leveraging a range query processing protocol. The key idea is that the cloud can launch a range query in which the center of the range is the query point q. Then, by gradually increasing the range scopes in a series of range queries, the cloud can gradually search for the nearby points from the dataset. Fig. 2 shows a simplified example. Assume that there are 8 1-D points stored on the cloud, i.e., $\{0, 1, \cdots, 7\}$. Given a query point $q = 3.5$ and 4NN query, the cloud first launches a range query $[3, 4]$ and gets two points $\{3, 4\}$. Then, the cloud has a larger range query $[2, 5]$ and gets another two points $\{2, 5\}$. Last, the cloud returns $\{2, 3, 4, 5\}$ as the 4NN query results.

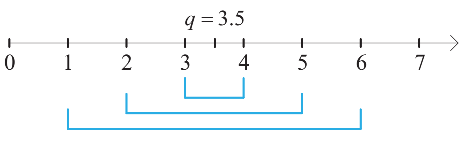
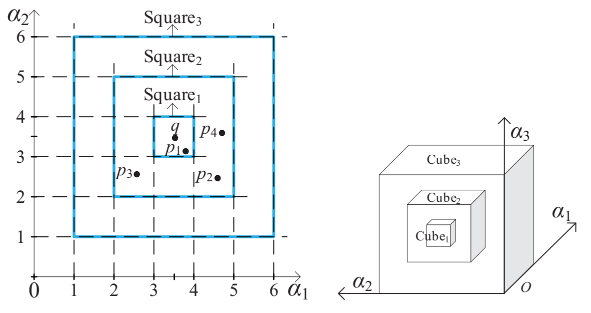


Fig. 2. kNN query processing in 1-D data.

2) 2-D Data: An approximate kNN query processing protocol in 2-D data can also be constructed by employing a range query processing protocol. An example is plotted in Fig. 3(a) to explain the idea. Two vectors $\overrightarrow{\alpha_1}$ and $\overrightarrow{\alpha_2}$ are randomly generated where $\overrightarrow{\alpha_1} \perp \overrightarrow{\alpha_2}$. The coordinate values of a point are defined by the scalar projection of the point onto the vectors $\overrightarrow{\alpha}_1$ and $\overrightarrow{\alpha}_2$, respectively. Given a query point $q = (3.5, 3.5)$ and 4NN query, the cloud first launches a range query $[3, 4]$ for all the points' scalar projection onto the vector $\overrightarrow{\alpha_1}$, i.e., $\text{Proj}_{\overrightarrow{\alpha_1}}(p_i) = \frac{\overrightarrow{p_i} \cdot \overrightarrow{\alpha_1}}{|\overrightarrow{\alpha_1}|}, i = [n]$, where $[n]$ is the abbreviation of $1, \cdots, n$. Let $R(\overrightarrow{\alpha_1})$ represent the corresponding query result set. Similarly, the cloud continues to have a range query of $[3, 4]$ for all the points' scalar projection onto the vector $\overrightarrow{\alpha_2}$, i.e., $\text{Proj}_{\overrightarrow{\alpha_2}}(p_i) = \frac{\overrightarrow{p_i} \cdot \overrightarrow{\alpha_2}}{|\overrightarrow{\alpha_2}|}, i = [n]$. Let $R(\overrightarrow{\alpha_2})$ denote the corresponding query result set. Then, the cloud computes the conjunctive result set of the above two range queries, i.e., $R(\overrightarrow{\alpha_1}) \cap R(\overrightarrow{\alpha_2})$. In Fig. 3(a), $R(\overrightarrow{\alpha_1}) \cap R(\overrightarrow{\alpha_2})$ contains all of the points in Square_1. In the example, the cloud gets one point p_1. Next, the cloud enlarges the range query to be $[2, 5]$ and finds all of the points in Square_2, in which the cloud gets points p_2, p_3, and p_4. Last, the cloud returns $\{p_1, p_2, p_3, p_4\}$ as the 4NN query results. In case more than 4 nearby points are required, (i.e., $k > 4$), the cloud continues to enlarge the range query to be $[1, 6]$ and searches for the points in Square_3. In this way, the cloud gradually increases the range scopes in the range queries and stops searching until k distinct points are found. Note that the proposed kNN protocol only returns approximate results. A strategy to ensure the query results accuracy is presented in Section II-B.



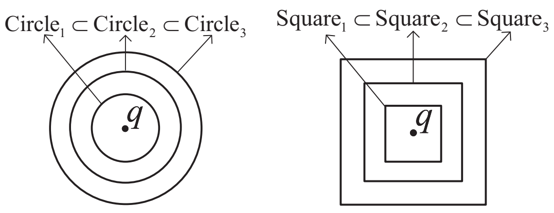
(a) kNN in 2-D data. (b) kNN in 3-D data.

Fig. 3. Approximate kNN query processing in 2-D data and 3-D data.

3) 3-D Data and Higher Dimensional Data: Note that a range query processing protocol can also be exploited to build an approximate kNN query processing protocol in 3-D data and higher-dimensional data. As shown in Fig. 3(b), by having 3 range queries onto three vectors $\overrightarrow{\alpha_1}$, $\overrightarrow{\alpha_2}$, $\overrightarrow{\alpha_3}$ in 3-D space, where $\overrightarrow{\alpha_1} \perp \overrightarrow{\alpha_2} \perp \overrightarrow{\alpha_3}$, and computing the conjunctive query results set, the cloud can search from the smallest Cube_1 to the larger cubes (e.g., Cube_2, Cube_3). The search process terminates until the cloud finds k distinct points. For higher-dimensional data (e.g., 4-D data), the cloud can launch multiple (e.g., 4) range queries to specify a higher dimensional cube (i.e., a hypercube). Likewise, the cloud can search from the smallest hypercube to the larger hypercubes and gradually finds k distinct points. In the rest of the paper, we focus on kNN protocol in 2-D data since 2-D location privacy is the major concern of this paper.

B. Accuracy-Assured Strategy

We now develop a strategy to ensure results accuracy for 2-D kNN protocol. As shown in Fig. 4, the accurate kNN search process should search nearby points in circles whose center is the query point q. However, the 2 range queries-based kNN protocol searches nearby points in squares. Therefore, it cannot guarantee the results accuracy.


Circle$_1$ ⊂ Circle$_2$ ⊂ Circle$_3$
q
Square$_1$ ⊂ Square$_2$ ⊂ Square$_3$
q

(a) Accurate kNN search. (b) Approximate kNN search.

Fig. 4. Accurate vs. approximate kNN search process.

To facilitate the description of the accuracy-assured strategy, we first describe some mathematical notations below. Consider that there are L levels of gradually increased range queries which specifies Square$_1, \cdots,$ Square$_L$, where Square$_1 \subset \cdots \subset$ Square$_L$. For each data point returned from the cloud, the data user can compute its Euclidean distance to the query point q. Let $dist(p, q)$ represent the Euclidean distance between two points p and q. As depicted in Fig. 5, for a point $p_i \in$ Square$_j$, if $dist(p_i, q) \leq r_j$, then p_i locates in Circle$_j$; otherwise, it locates in Square$_j$ but outside of Circle$_j$. Let all points in Square$_j$ that satisfy $d(p_i, q) \leq r_j$ be in the *accuracy-assured set*, denoted as $\mathcal{A}_j$. Moreover, let all points in Square$_j$ that satisfy $d(p_i, q) > r_j$ be in the *accuracy-uncertainty set*, denoted as $\mathcal{U}_j$. We define $R_j = \mathcal{A}_j \cup \mathcal{U}_j$, so R_j contains all points in Square$_j$. We use $|\mathcal{A}_j|$, $|\mathcal{U}_j|$, and $|R_j|$ to denote the cardinality of the set $\mathcal{A}_i$, $\mathcal{U}_i$, and R_j, respectively.

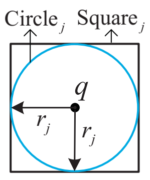


Fig. 5. Points in Circle$_j$ are in $\mathcal{A}_j$. Points outside Circle$_j$ but in Square$_j$ are in $\mathcal{U}_j$.

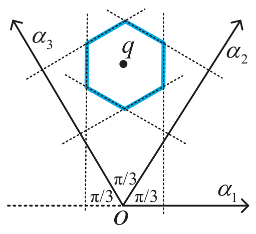


Fig. 6. The 3 range queries onto 3 vectors specify a regular hexagon.

Our accuracy-assured strategy is developed based on the following critical observation: ***all points in $\mathcal{A}_i$ must be the accurate top $|\mathcal{A}_i|$NN results. The top k ($k \leq |\mathcal{A}_i|$) nearest points in $\mathcal{A}_i$ must be the accurate top kNN results.***

More specifically, the accuracy-assured strategy is described in Algorithm 1. In the algorithm, the cloud first searches in Square$_1$ and obtains the result set R_1. If $|R_1| < \varepsilon k$ (the result set expansion factor $\varepsilon > 1$), then the cloud continues to search for larger squares until $|R_j| \geq \varepsilon k$. The adjustable factor ε is introduced to ensure a slightly more than k points are returned to the data user. Assume that the cloud finds sufficient points after searching of Square$_j$, then the cloud returns all points in Square$_j$ to the data user. The data user can identify all the data points that in the accuracy-assured set $\mathcal{A}_j$. If $|\mathcal{A}_j| < k$, the data user informs the cloud to continue to search in larger squares until the data user can get a set $\mathcal{A}_j$ such that $|\mathcal{A}_j| \geq k$. Then, the data user sorts the data points in $\mathcal{A}_j$ according to their distance to the query point q and selects the top k nearest points as the accurate kNN query results. Note that by properly choosing the expansion factor ε, this strategy only requires a small constant number of interaction rounds between the cloud and the data user.

Algorithm 1: Accuracy-assured strategy

```
Input: $p_i (i = [n])$; $q$; $k$; $\varepsilon$; $L$ levels of gradually increased
       range queries which specifies Square$_1, \cdots,$ Square$_L$.
Output: Accurate $k$NN query results.
1  Initialization: $R_j = \emptyset$, for $j = 0, \cdots, L$; $j = 0$;
2  while ($|R_j| < \varepsilon k$) do
3      $j++$;
4      The cloud searches in Square$_j$ and obtains the result set
         $R_j$;
5  The cloud returns $R_j$ to the data user for computing $|\mathcal{A}_j|$;
6  while ($|\mathcal{A}_j| < k$) do
7      $j++$;
8      The cloud continues to search in Square$_j$ and obtains the
         result set $R_j$;
9      The cloud returns $R_j$ to the data user for computing $|\mathcal{A}_j|$;
10 The data user sorts the data points in $\mathcal{A}_j$ according to their
     distance to the query point $q$ and selects the top $k$ nearest
     points as the accurate $k$NN query results;
```

C. Extending to Multiple Vectors

In the 2-D kNN protocol, more than 2 vectors can be used so that the cloud can have range queries onto each of them in searching. It is observed that the more vectors, the smaller $|\mathcal{U}_j|$. For example, consider there are 3 vectors $\overrightarrow{\alpha}_1$, $\overrightarrow{\alpha}_2$, and $\overrightarrow{\alpha}_3$ that equally divide 2-D space (i.e., the degree between two adjacent vectors is $\pi/3$). As shown in Fig. 6, the corresponding 3 range queries specify a regular hexagon. Generally speaking, d vectors specify a regular polygon with $2d$ edges. As exhibited in Fig. 7, the larger d is, the closer the search area is a circle and the smaller $|\mathcal{U}_j|$. In the extreme case, we have $\lim_{d \to +\infty} |\mathcal{U}_j| = 0$. Then, the kNN search is exactly in circles and the returned results are always accurate.

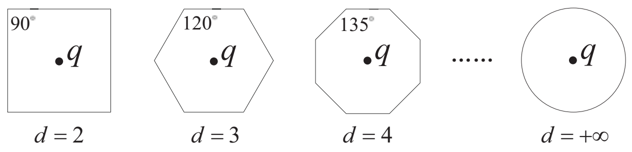


Fig. 7. Geometrically, the search area is a regular polygon with $2d$ edges. The larger d is, the closer the search area is a circle.

III. Secure kNN Protocol

This section describes how to convert the above kNN protocol to be a secure kNN protocol. We first describe how to construct a range query processing protocol from keyword queries. Then, we introduce how to differentiate range queries onto multiple vectors. Next, the secure index data structure is elaborated, followed by the detailed secure kNN protocol design. Last, the secure analysis of FSkNN is presented.

A. Prefix-based Range Query Processing Protocol

Theoretically, any existing secure range query protocols (e.g., [15]) can be exploited to build a secure kNN protocol. In this paper, we employ the prefix membership verification scheme [16] to design a range query processing protocol based on keyword queries. For an integer p of v bit, its binary representation can be represented as $b_1 b_2 \cdots b_v$. The *prefix family* of the number p, denoted as $F(p)$, is set of $v+1$ prefixes $\{b_1 b_2 \cdots b_v, b_1 b_2 \cdots b_{v-1}*, \cdots, b_1 * \cdots *, * * \cdots *\}$. For instance, the prefix family of integer 3 of 3 bits is $F(3) = F(011) = \{011, 01*, 0**, ***\}$. For a range $[a, b]$, it can be represented by a minimum set of prefixes (called *minimum prefix family*), denoted as $S([a, b])$, such that the union of the prefixes equal to $[a, b]$. For example, $S([0, 4]) = S([000, 100]) = \{0**, 100\}$, where the prefix "$0**$" represents $0, 1, 2, 3$ and the prefix "100" represents 4. Therefore, the union of the prefixes "$0**$" and "100" equal to all integers in the range $[000, 100]$. There is an important property holds between the prefix family of an integer p and the minimum prefix family of a range $[a, b]$. ***Given an integer p and a range $[a, b]$, $p \in [a, b]$ if and only if there exists a prefix $pf \in S([a, b])$ such that $pf \in F(p)$. Or equivalently, $p \in [a, b]$ if and only if $F(p) \cap S([a, b]) \neq \emptyset$.*** An example is presented below to explain this property.

Example. To test if the range [0, 4] contains the integer 3, we first compute the minimum prefix family of the range [0, 4] as $S([0, 4]) = S([000, 100]) = \{0**, 100\}$. Then, we compute the prefix family of the integer 3 of 3 bits as $F(3) = F(011) = \{011, 01*, 0**, ***\}$. Because $S([0, 4]) \cap F(3) = \{0**\} \neq \emptyset$, it holds that $3 \in [0, 4]$.

Based on the above property, the prefix-based range query processing protocol can be constructed based on disjunctive keyword queries. Given n 1-D points $p_1, \cdots, p_n$, the data owner computes the prefix families $F(p_1), \cdots F(p_n)$ and outsources them to store on the cloud. For a range query $[a, b]$, the data user computes $S([a, b])$ and sends it to the cloud. On receiving $S([a, b])$, for a point p_i, the cloud checks if any prefix in $S([a, b])$ is also in $F(p_i)$. If there exists any prefix in $S([a, b])$ that is also in $F(p_i)$, then the cloud puts p_i in the range query result set. Let $R(pf_i)$ represent the query result set for a prefix $pf_i \in S[a, b]$, then the final range query result set can be computed by the disjunctive operation, i.e., $\bigcup_{i=1}^{|S[a,b]|} R(pf_i)$. Hence, by treating each prefix as a keyword, the prefix-based range query protocol can be constructed using disjunctive keyword queries.

B. Range Queries onto Multiple Vectors

The kNN protocol in 2-D data leverages range queries onto multiple vectors. To distinguish the range queries on different vectors, we develop the following strategy. Assume that the data owner intends to calculate the prefix family of a point p_i onto vector $\overrightarrow{\alpha_j}$, denoted as $F(p_i, \overrightarrow{\alpha_j})$. To make each prefix family unique, the data owner lets each prefix in a prefix family append its projected vector. For example, assume that $\text{Proj}_{\overrightarrow{\alpha_j}}(p_i) = 3$, then we have $F(p_i, \overrightarrow{\alpha_j}) = F(3, \overrightarrow{\alpha_j}) = \{011||\overrightarrow{\alpha_j}, 01*||\overrightarrow{\alpha_j}, 0**||\overrightarrow{\alpha_j}, ***||\overrightarrow{\alpha_j}\}$, where "$||$" represents concatenation. Then, the data owner outsources $F(3, \overrightarrow{\alpha_j})$ to the cloud for storage. For a range query $[a, b]$ onto vector $\overrightarrow{\alpha_j}$, the data user also needs to append the projected vector to each prefix in its minimum prefix family before sending to the cloud for query results. Let $S([a, b], \overrightarrow{\alpha_j})$ represent the minimum prefix family of $[a, b]$ onto vector $\overrightarrow{\alpha_j}$. For instance, $S([0, 4], \overrightarrow{\alpha_j}) = \{0**||\overrightarrow{\alpha_j}, 100||\overrightarrow{\alpha_j}\}$. Because $F(3, \overrightarrow{\alpha_j}) \cap S([0, 4], \overrightarrow{\alpha_j}) = \{0**||\overrightarrow{\alpha_j}\}$, we have $3 \in [0, 4]$ onto vector $\overrightarrow{\alpha_j}$.

C. Secure Index Design

We have shown that a kNN protocol can be constructed from conjunctive range queries and a range query protocol can be constructed from disjunctive keyword queries. Therefore, ***a kNN protocol can be constructed based on conjunctive and disjunctive keyword queries.*** The secure index used in FSkNN for the underlying keyword queries is built based on a data structure called the random Bloom filter (RBF).

RBF. The data structure RBF [17] is revised from the classic Bloom filter [18]. A set containing multiple keywords can be inserted into an RBF. After insertion, the RBF can be used to check if a keyword is inserted or not. As shown in Fig. 8, an RBF can be viewed as a 2-D binary array B with 2 rows and m columns. Let $B[i][j] \in \{0, 1\}$ denote the bit value in the ith row and jth column of an RBF B. Each entry in the 2-D array is called a cell. The two cells in a column are called a cell-pair. For an empty RBF, each cell in each cell-pair is initialized to be either 0 or 1 randomly while keeping the bit values of two cells in a cell-pair to be different. According to their bit values, the two cells in a cell-pair are called 0-cell and 1-cell, respectively. An RBF is associated with a random number r_B, $k_B + 1$ hash functions h_1, h_2, $\cdots$, h_{k_B+1}, and a random oracle H [19]. In this work, we use the keyed hash message authentication code (HMAC) [19] to instantiate these hash functions. Suppose that there are $k_B + 1$ secret keys $k_1, \cdots, k_B + 1$. For the first k_B hash functions, they can be represented as $h_i(\cdot) = \text{HMAC}_i(\cdot) \mod m$, for $i = [k_B]$. For the last hash function, it can be denoted as $h_{k_B+1}(\cdot) = \text{HMAC}_{k_B+1}(\cdot)$. The used random oracle is instantiated as $H(\cdot) = \text{SHA256}(\cdot) \mod 2$. Let $\oplus$ represent XOR operation.

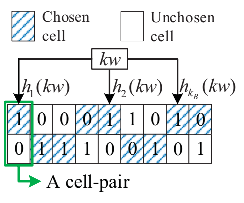


Fig. 8. An RBF example.

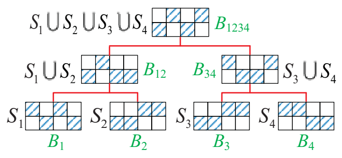


Fig. 9. An RBF tree example.

To insert a keyword kw into an RBF, for $j = [k_B]$, we set

$$B[H(h_{k_B+1}(h_j(kw)) \oplus r_B)][h_j(kw)] = 1. \quad (1)$$

In Eq. (1), $H(h_{k_B+1}(h_j(kw)) \oplus r_B)$ determines the *chosen cell* from a cell-pair. The other *unchosen cell* of a cell-pair is set to be 0, i.e., for $j = [k_B]$, we set

$$B[1-H(h_{k_B+1}(h_j(kw)) \oplus r_B)][h_j(kw)]=0. \qquad (2)$$

Note that the chosen cell for a given cell-pair is always the same. For example, in inserting two different keywords kw_1 and kw_2, suppose that two different keywords kw_1 and kw_2 are mapped to the same cell-pair under two different hash functions h_1 and h_2, respectively. That is, $h_1(kw_1) = h_2(kw_2)$. Then, it holds that $H(h_{k_B+1}(h_1(kw_1)) \oplus r_B) = H(h_{k_B+1}(h_2(kw_2)) \oplus r_B)$. Therefore, the chosen cell for $h_1(kw_1)$ (or $h_2(kw_2)$)-th cell-pair always stays the same during the keywords insertion.

To check whether a keyword kw is inserted into the RBF B, we need to re-compute Eq. (1) and test whether this equation holds for all $j \in [k_B]$. If it holds, then the keyword kw is inserted into the RBF, otherwise not.

For the traditional Bloom filter, the more keywords are inserted, the more 1-cells appear in the Bloom filter. Hence, the information about the number of embedded keywords is partially leaked. ***However, the RBF-based index is a completely random data structure consists of an equal number of 1-cells and 0-cells. Therefore, the RBF-based index is more secure than the traditional BF-based index.***

RBF Tree. To achieve sublinear time for a keyword search, we organize multiple RBFs to be a balanced binary tree structure. An example of the RBF tree is depicted in Fig. 9. Consider we want to build an RBF tree from 2 RBFs B_l and B_r, where B_l is the left child RBF and B_r is the right child RBF. Let B_p represent their parent RBF and $\vee$ represent logical OR operation. For $i = [m]$, B_p is computed as

$$\begin{aligned} &B_p[H(h_{k_B+1}(i) \oplus r_{B_p})][i] = B_l[H(h_{k_B+1}(i) \oplus r_{B_l})][i] \\ &\vee B_r[H(h_{k_B+1}(i) \oplus r_{B_r})][i]. \end{aligned} \qquad (3)$$

Eq. (3) shows that B_p's ith cell-pair is the bitwise OR operation of B_l's ith cell-pair and B_r's ith cell-pair. If there are more than 2 RBFs, we can repeatedly use the above approach to generate a balanced binary RBF tree from a group of leaf RBFs in a bottom-up manner until there is only one root RBF. There is an important property holds between a parent RBF B_p and its two children RBFs B_l and B_r: ***if RBF B_l embeds a keyword set S_1 and RBF B_r embeds a keyword set S_2, then their parent RBF B_p embeds the keyword set $S_1 \cup S_2$.*** Below is an example to explain this property.

Example. Suppose that RBFs B_l and B_r have 6 cell-pairs. The chosen cells of RBFs B_l and B_r constitute the bit arrays $A_l = 001010$ and $A_r = 101000$, respectively. According to Eq. (3), the chosen cells of the parent RBF B_p is a bit array A_p computed by $A_p = 001010 \vee 101000$ (bitwise OR) $= 101010$. Suppose that in inserting a keyword kw into RBF B_l, it is mapped to the 3rd and 5th locations, so we have $A_l = 001010$. Due to the bitwise OR operation, the 3rd and 5th locations in the bit array A_p must be 1. Therefore, the keyword kw must also be inserted in the RBF B_p. The random oracle $H(\cdot)$ in Eq. (1)-(3) is introduced to keep the number of 1-cells and 0-cells equal. In the presence of the random oracle, the above property still holds. Consequently, if RBF B_l embeds a keyword set S_1 and RBF B_r embeds a keyword set S_2, then their parent RBF B_p embeds the keyword set $S_1 \cup S_2$.

The above property allows us to search from the root RBF to a leaf RBF in $O(\log n)$ time. Assume that there are four RBFs $B_1, \cdots, B_4$, where B_i embeds the keyword set S_i, for $i = [4]$. The four RBFs generate a binary RBF tree as shown in Fig. 9. Assume that a keyword $kw \in S_2$ but $kw \notin S_1, S_3, S_4$. To search kw in the RBF tree, the cloud first searches in the root RBF B_{1234} and finds that kw is inserted in it (because $kw \in S_1 \cup S_2 \cup S_3 \cup S_4$). Then, the cloud continues to search the left child RBF B_{12} and right child RBF B_{34}. The cloud finds that B_{34} does not embed kw, so the search terminates (all RBFs under B_{34} are pruned). Next, the cloud finds that B_{12} embeds kw (because $kw \in S_1 \cup S_2$), so the cloud continues to search the left child RBF B_1 and right child RBF B_2. Last, the cloud confirms that kw is inserted in B_2 and hence $kw \in S_2$. Thus, it only takes $O(\log n)$ time to perform a keyword search over an RBF tree.

D. SkNN Protocol Design

We next introduce the detailed FSkNN design. FSkNN is composed of four subroutines: (1) index-building, (2) token-generation, (3) query-processing, and (4) post-processing.

(1) **Index-building.** For each data point in the database, the data owner computes the prefix families for its scaler projection onto each vector, i.e., $F(p_i, \overrightarrow{\alpha_j})$, for $i = [n], j = [d]$. All prefixes of a point are embedded into a distinct RBF. Each RBF represents a point and is associated with its encrypted data item by using a pointer (see Fig. 1). All RBFs generated by data points now serve as the leaf nodes to construct a balanced binary RBF tree (see Eq. (3)). Note that each RBF in the RBF tree is associated with a random number. The secure index is the RBF tree and its RBF-specific random numbers.

(2) **Token-generation.** Given a 2-D query point $q = (x, y)$, the data user chooses L levels of ranges radiuses r_i $(i = [L])$ where $r_1 < r_2 < \cdots < r_L$. For a vector $\overrightarrow{\alpha_j}$, the data user computes $\text{Proj}_{\overrightarrow{\alpha_j}}(q)$. Then, the data user computes the minimum prefix family of each range scope onto each vector. To facilitate description, we define $Q(j, i, q) = S([\text{Proj}_{\overrightarrow{\alpha_j}}(q) - r_i, \text{Proj}_{\overrightarrow{\alpha_j}}(q) + r_i], \overrightarrow{\alpha_j})$. The data user computes $Q(j, i, q)$ for $j = [d], i = [L]$. For each prefix in $Q(j, i, q)$, the data user treats it as a keyword. For a keyword kw, the data user computes k_B-pair of hashes and locations: $\{h_{k_B+1}(h_{i'}(kw)), h_{i'}(kw)\}$, for $i' = [k_B]$. ***The k_B-pair of hashes and locations serve as the search token t_{kw} of keyword kw.*** The data user generates search tokens in the above way for all prefixes in $Q(j, i, q)$ $(j = [d], i = [L])$ and sends these search tokens to the cloud for search results.

(3) **Query-processing.** On receiving these search tokens, the cloud processes them one by one in sequential. Let t_{kw} represent the search token for a keyword kw. The search process is described below. Let $t_{kw}[i]$ represent the ith ordered pair in t_{kw}. We have $t_{kw}[i] = \{h_{k_B+1}(h_i(kw)), h_i(kw)\}$. Let $t_{kw}[i][1]$ and $t_{kw}[i][2]$ represent the first and second element

in $t_{kw}[i]$, respectively. To check if kw is inserted in an RBF B, the cloud tests if there exist $i \in [k_B]$ such that $B[H(t_{kw}[i][1] \oplus r_B)][t_{kw}[i][2]] = 0$. If yes, then it holds that kw is not inserted in the RBF B. If the cloud finds that kw is not inserted in the RBF B, then the cloud stops searching (all RBFs under RBF B are pruned). Otherwise, the cloud tests if kw is inserted in the left and right children of the RBF B. The binary search tops until the cloud arrives at the leaf RBF and gets the result. ***Harnessing the above process for a secure keyword search, the cloud can have disjunctive keyword queries (by treating each prefix as a keyword) to realize the secure range query (see Section III-A). Based on the secure range query, the cloud can realize secure kNN search (see Section II).*** Once enough points are obtained, the cloud stops searching. Then, the cloud returns the searched encrypted data items to the data user.

(4) **Post-processing.** On receiving the encrypted data items from the cloud, the data user exploits the results accuracy-assured strategy (see Algorithm 1) to obtain the accurate kNN query results.

E. Security Analysis

FSkNN is built based on using conjunctive range queries. The adopted prefix-based range query is realized by using disjunctive keyword queries (each prefix is treated as a keyword). Hence, FSkNN is constructed based on conjunctive and disjunctive keyword queries. The security of the underlying keyword query protocol can provide the security guarantee for FSkNN. Therefore, we prove the security of the underlying keyword query protocol in this section.

Secure Model & Leakage Function. To prove the security of a keyword query protocol, we adopt the widely used secure model called adaptive Indistinguishability under Chosen-Keyword Attack (IND-CKA) [20]. The set of data items is denoted as $\mathbf{D} = \{d_1, \cdots, d_n\}$. Let $\mathbb{I}$, $\mathbf{T}$, and $\mathbf{c}$ represent the real index, real search token, and real ciphertext, respectively. We assume that a CPA-secure encryption algorithm [19] is adopted by FSkNN to encrypt data items. Two leakage functions $\mathcal{L}_1$ and $\mathcal{L}_2$ specify the information that is allowed to be leaked to the adversary. (1) $\mathcal{L}_1(\mathbb{I}, \mathbf{D})$: On input $\mathbb{I}$ and $\mathbf{D}$, it reveals the size of each RBF, the number of data items in the database, the data item identifiers $ID = (id_1, \cdots, id_n)$, and the size of each encrypted data item. (2) $\mathcal{L}_2(\mathbb{I}, \mathbf{D}, kw)$: On input $\mathbb{I}$, $\mathbf{D}$, and kw, it reveals the *access pattern* and the *search pattern*. The access pattern shows the data item identifiers that match the keyword query kw. The search pattern shows whether the same search token was searched before or not.

Secure Intuition. The intuition for data privacy is that the data items are formally encrypted. The intuition for index privacy is that the RBF tree-based index is a random data structure with an equal number of 1-cells and 0-cells. The intuition for token privacy is that the search tokens are generated by non-invertible one-way hashes. To sum up, it is hard for the cloud to infer the useful information from just viewing the encrypted data items, RBF-tree based index, and hash-generated search tokens. Formally, we have the following Theorem.

Theorem 1: Under the permitted leakage functions $\mathcal{L}_1$ and $\mathcal{L}_2$, FSkNN scheme is adaptive IND-CKA secure in the random oracle model.

We exploit the simulation proof technique [21] to prove the security of FSkNN. ***If a probabilistic polynomial-time (PPT) adversary cannot extract any useful information from a simulated view to gain non-negligible advantages to effectively distinguish between a simulated view and the real view, then FSkNN is secure.*** The simulation proof thus captures the notion that nearly "nothing" is learned beyond the permitted information leakage. Let $\mathbb{I}^*$, $\mathbf{T}^*$, and $\mathbf{c}^*$ represent the simulated index, simulated search token, and simulated ciphertext, respectively. In the proof, we first describe an efficient simulator $\mathcal{S}$ that can simulate a view $A_v^* = (\mathbb{I}^*, \mathbf{T}^*, \mathbf{c}^*)$ with the help of information accessible in the leakage functions $\mathcal{L}_1$ and $\mathcal{L}_2$. Next, we show that a PPT adversary cannot distinguish between the simulated view $A_v^* = (\mathbb{I}^*, \mathbf{T}^*, \mathbf{c}^*)$ and the real adversary view $A_v = (\mathbb{I}, \mathbf{T}, \mathbf{c})$.

Proof: (1) **For data privacy** (simulate $\mathbf{c}^*$). To simulate an encrypted data item, $\mathcal{S}$ can obtain the size of each encrypted data item from the leakage function $\mathcal{L}_1$. Then, $\mathcal{S}$ selects any plaintext and encrypts it by using a CPA-secure encryption algorithm. The encrypted ciphertext serves as the simulated ciphertext. Note that $\mathcal{S}$ must ensure that the simulated ciphertext has the same size as the real ciphertext. Next, $\mathcal{S}$ continues to use the above approach to simulate all encrypted data items. Because the CPA-secure encryption algorithm achieves ciphertext indistinguishability, a PPT adversary cannot distinguish the simulated ciphertext with the real ciphertext.

(2) **For index privacy** (simulate $\mathbb{I}^*$). To simulate the secure index, $\mathcal{S}$ first constructs an RBF tree T with an identical structure with the real index $\mathbb{I}$. $\mathcal{S}$ ensures that each RBF in T has the same size as the RBF in $\mathbb{I}$. At first, for each cell-pair in an RBF in T, how to assign 0-cell and 1-cell is determined randomly. Next, a random number is generated by $\mathcal{S}$ to associate with each RBF in T. Finally, $\mathcal{S}$ uses T and its associated random number as the simulated index $\mathbb{I}^*$. As a result, $\mathbb{I}^*$ and $\mathbb{I}$ are indistinguishable for a PPT adversary.

(3) **For token privacy** (simulate $\mathbf{T}^*$). Assume that $\mathcal{S}$ receives a keyword query kw. From $\mathcal{L}_2$, $\mathcal{S}$ knows whether this query has been searched before or not. If it is searched before, $\mathcal{S}$ re-uses the previous simulated token t_{kw}. Otherwise, $\mathcal{S}$ needs to generate a new search token as illustrated below. As mentioned above, the search token consists of k_B-pair of hashes and locations. From $\mathcal{L}_2$, $\mathcal{S}$ knows the information about which leaf RBF in the index contains the keyword kw. If an RBF contains the keyword kw, then we say that the RBF matches the search token t_{kw}. For the leaf RBF that matches t_{kw}, $\mathcal{S}$ can program the output bit of the random oracle H to select k_B-pair of hashes and locations to generate t_{kw} while ensuring that the RBF matches the simulated t_{kw}. For the leaf RBF that does not match t_{kw}, $\mathcal{S}$ can program the output bit of H to generate t_{kw} while ensuring that the RBF does not match the simulated t_{kw}. By this way, the simulated search token $\mathbf{T}^*$ can be generated. Obviously, a PPT adversary cannot

TABLE II
COMPARISON BETWEEN TWO PREVIOUS SCHEMES AND FSKNN (NA REPRESENTS "NOT APPLICABLE").

Scheme	**Time cost**						**Space cost**				**Ave. interaction rounds**
	$k=1$			$k=50$			Token size	Index size			
	n=10^4	n=10^5	n=10^6	n=10^4	n=10^5	n=10^6		n=10^4	n=10^5	n=10^6	
FSkNN	23 ms	46 ms	73 ms	61 ms	94 ms	148 ms	0.43 MB	0.57 GB	4.07 GB	20.5 GB	1.09
Yao et al. [9]	5 ms	9 ms	12 ms	Na	Na	Na	8 byte	14.8 MB	17.4 MB	20.3 MB	1
Elmehdwi et al. [7]	0.15 s	1.44 s	14.3 s	0.12 min	1.18 min	11.8 min	16 byte	Na	Na	Na	1

distinguish between $\mathbf{T}^*$ and $\mathbf{T}$ because the search token is produced by the random hash functions and the random oracle.

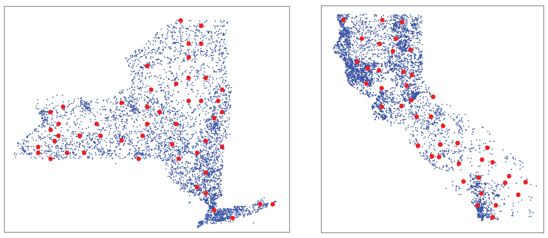
In summary, from the view of a PPT adversary, $A_v^* = (\mathbb{I}^*, \mathbf{T}^*, \mathbf{c}^*)$ and $A_v = (\mathbb{I}, \mathbf{T}, \mathbf{c})$ are indistinguishable. Hence, under the permitted leakage functions $\mathcal{L}_1$ and $\mathcal{L}_2$, FSkNN scheme is adaptive IND-CKA secure in the random oracle model. ■

IV. EXPERIMENTS

In this section, we evaluate the performance of FSkNN and compare it with two previous schemes.

A. Experiment Setup

C++ is used to implement FSkNN. A machine (128 GB RAM and 2.5 GHz dual CPUs) is leveraged to work as the cloud. We set the gap between two consecutive range radiuses to be fixed. Let $\delta = r_1 = r_i - r_{i-1}$, for $i = 2, \cdots, L$. We set $\delta = 0.02 \max\{d(p_i, p_j)\}$, for $i = [n], j = [n]$. The default value of the number of data items n is set to be 10^5. The number of points needed in a query k is set to be 50. The result set expansion factor ε is set to be 1.8. The number of used vectors d is set to be 3. The number of hash functions used by the RBF k_B is set to be 7. The number of ranges scopes L is set to be 8. We adjust the number of cell-pairs in the RBF so that the false positive rate of the secure index is extremely small and we cannot observe any false positive instances. In our experiments, when we change the value of one parameter under consideration, other parameters are kept at their default values.

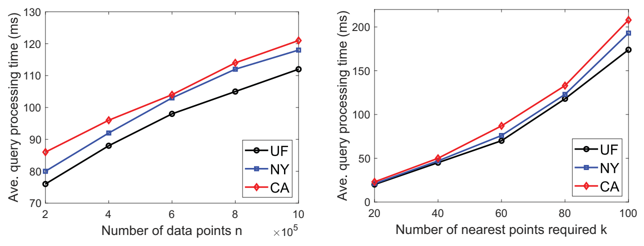


(a) New York (NY) (b) California (CA)

Fig. 10. The data distribution of two real-world datasets.

Three different datasets are used in the experiment. We download two real-world datasets with 1 million points from a crowdsourcing project: OpenStreetMap [22]. The first one is collected from the state of New York, so it is represented as **NY**. The second one is collected from California, denoted as **CA**. The third dataset is a synthetic dataset generated from the uniform distribution, represented as **UF**. Fig. 10 depicts the data distributions of two real-world datasets. Three performance metrics are tested: (1) time cost, (2) space cost, and (3) interaction rounds. The time cost is the total time needed by the cloud to process an SkNN search. The space cost consists of the search token size and the secure index size stored by the cloud. The interaction rounds represent the number of user-cloud interactions in an SkNN search.

B. Experimental Results



(a) Time v.s. n ($k = 50$). (b) Time v.s. k ($n = 10^5$).

Fig. 11. Average query processing time when changing n and k.

We report the time cost, space cost, and iteration rounds of FSkNN as follows. (1) **Time Cost.** The average query processing time (when changing n and k) is shown in Fig. 11. With an increasing n, the average query processing time increases sublinearly. With an increasing k, the average query processing time increases slightly faster than linear. For $k = 50$, it takes less than 130 ms for a secure 50NN search in a dataset containing 1 million data items. (2) **Space Cost.** For the space cost, the secure search token size and the secure index size are measured. Table II shows the token size for an SkNN query in FSkNN. The search token of FSkNN only needs 0.43 MB, indicating a low communication volume in transmission. For the index size, the existing Bloom filter compression algorithms [17] is exploited to compress the index. As exhibited in Table II, the secure index takes 20.5 GB space for 10^6 data items. (3) **Interaction Rounds.** The experiments show that the average interaction rounds are monotonically decreasing with an increasing expansion factor ε. When the expansion factor ε grows from 1 to 2, the average interaction rounds decrease from 2.13 to 1.05. When $\varepsilon = 1.8$, the average rounds are only 1.09, which means that in the vast majority of cases, only one interaction round is required. Thus, FSkNN only has a small constant number of interaction rounds.

C. Comparison

We compare FSkNN with two previous secure kNN schemes: Yao's scheme [9] and Elmehdwi's scheme [7]. Table II summaries the comparison results. We have four observations. (1) both FSkNN and Yao's schemes have very fast query processing time (less than 1 second). (2) all three schemes do not have very large search tokens (less than 0.5 MB). (3) the average interaction rounds for all of the three schemes are small (less than 1.1). (4) FSkNN's index is significantly larger than the other two schemes. ***In essence, this is because FSkNN trades space (used for index storage) for higher security, faster query process, and results accuracy. Such tradeoff is reasonable because space is an abundant and cheap resource for a powerful cloud.*** For example, Google Drive storage price is 100 GB for $1.99 per month [23]. Tens of GB storage in FSkNN is affordable for an LSP. However, other metrics including higher security, faster query process, and results accuracy are much more crucial for good user experience in real-world applications.

V. Related Work

The existing techniques for SkNN query processing can be roughly divided into the following six categories. The first type of works use location obfuscation [24] and data transformation approach [25], [26]. These works provide high query process efficiency, but they only provide very weak privacy guarantees since they do not use strong standard encryption algorithms. Second, fully homomorphic encryption (FHE) [5] allows cloud to have secure kNN search directly over the encrypted dataset. Nevertheless, the search performance of current FHE-based solutions is still not satisfactory. Third, the private information retrieval (PIR) technique is exploited in several works such as [27]. PIR-based schemes can support the data user's query location privacy, but they suffer from two limitations. (1) It does not consider how to protect data privacy. (2) PIR-based schemes have a very long query processing time (especially for big datasets). Forth, the Voronoi diagram approach is proposed in [9] for addressing secure 1NN search. This scheme requires each data user to store and maintain a big local index for query processing, which makes it not suitable for weak clients with limited storage capacity. Fifth, property-preserving encryption approach (including distance-recoverable encryption (DRE) and order-preserving encryption (OPE)) is developed in [8], [26], [28]. DRE-based and OPE-based schemes are quite efficient in searching, However, as analyzed in [13], they only achieve weak security since their encryption method leaks statistical information to the adversary. Sixth, a projection function-based solution is proposed in [10], but this solution fails to support accurate query results.

VI. Conclusion

In this paper, we come up with a brand new approach to realize fast and secure kNN query processing in cloud computing. The key contribution lies in: we show that a secure kNN protocol can always be constructed based on a secure range query protocol. Therefore, any future advances in the secure range query protocol can directly lead to advances in the secure kNN protocol.

References

[1] "General data protection regulation," https://eur-lex.europa.eu/eli/reg/2016/679/oj.

[2] G. Danezis, J. Domingo-Ferrer, M. Hansen, J.-H. Hoepman, D. L. Metayer, R. Tirtea, and S. Schiffner, "Privacy and data protection by design-from policy to engineering," *arXiv preprint arXiv:1501.03726*, 2015.

[3] X. Lei, X. Liao, T. Huang, H. Li, and C. Hu, "Outsourcing large matrix inversion computation to a public cloud," *TCC*, 2013.

[4] X. Lei, X. Liao, T. Huang, and H. Li, "Cloud computing service: The caseof large matrix determinant computation," *TSC*, pp. 688–700, 2015.

[5] C. Gentry, "Fully homomorphic encryption using ideal lattices." in *STOC*, 2009, pp. 169–178.

[6] R. Agrawal, J. Kiernan, R. Srikant, and Y. Xu, "Order preserving encryption for numeric data," in *SIGMOD*, 2004, pp. 563–574.

[7] Y. Elmehdwi, B. K. Samanthula, and W. Jiang, "Secure k-nearest neighbor query over encrypted data in outsourced environments," in *ICDE*, 2014, pp. 664–675.

[8] B. Wang, Y. Hou, and M. Li, "Practical and secure nearest neighbor search on encrypted large-scale data," in *INFOCOM*, 2016, pp. 1–9.

[9] B. Yao, F. Li, and X. Xiao, "Secure nearest neighbor revisited," in *ICDE*, 2013, pp. 733–744.

[10] X. Lei, A. X. Liu, R. Li, and G.-H. Tu, "Seceqp: A secure and efficient scheme for sknn query problem over encrypted geodata on cloud," in *ICDE*, 2019, pp. 662–673.

[11] S. Bugiel, S. Nurnberger, A. Sadeghi, and T. Schneider, "Twin clouds: An architecture for secure cloud computing," in *WCSC*, 2011.

[12] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *ICTACT*, 1999.

[13] M. Naveed, S. Kamara, and C. V. Wright, "Inference attacks on property-preserving encrypted databases," in *CCS*, 2015, pp. 644–655.

[14] A. Okabe, B. Boots, K. Sugihara, and S. N. Chiu, *Spatial tessellations: concepts and applications of Voronoi diagrams*. John Wiley & Sons, 2009.

[15] J. Wang and S. S. Chow, "Forward and backward-secure range-searchable symmetric encryption," IACR Cryptology ePrint Archive, Tech. Rep., 2019.

[16] A. X. Liu and F. Chen, "Collaborative enforcement of firewall policies in virtual private networks," in *PODC*, 2008, pp. 95–104.

[17] R. Li and A. X. Liu, "Adaptively secure conjunctive query processing over encrypted data for cloud computing," in *ICDE*, 2017, pp. 697–708.

[18] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Communications of the ACM*, 1970.

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

[20] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky, "Searchable symmetric encryption: improved definitions and efficient constructions," in *CCS*, 2006, pp. 79–88.

[21] Y. Lindell, "How to simulate it–a tutorial on the simulation proof technique," in *Tutorials on the Foundations of Cryptography*, pp. 277–346.

[22] "Openstreetmap," http://www.openstreetmap.org/.

[23] "Google drive paid consumer storage plans become google one," https://zd.net/2SpyU6x.

[24] M. F. Mokbel, C.-Y. Chow, and W. G. Aref, "The new casper: query processing for location services without compromising privacy," in *VLDB*, 2006, pp. 763–774.

[25] A. Khoshgozaran and C. Shahabi, "Blind evaluation of nearest neighbor queries using space transformation to preserve location privacy," in *SSTD*, 2007, pp. 239–257.

[26] W. K. Wong, D. W.-l. Cheung, B. Kao, and N. Mamoulis, "Secure knn computation on encrypted databases," in *SIGMOD*, 2009, pp. 139–152.

[27] X. Yi, R. Paulet, E. Bertino, and V. Varadharajan, "Practical k nearest neighbor queries with location privacy," in *ICDE*, 2014, pp. 640–651.

[28] H. Hu, J. Xu, C. Ren, and B. Choi, "Processing private queries over untrusted data cloud through privacy homomorphism," in *ICDE*, 2011, pp. 601–612.