Secure and Efficient Nearest Neighbor Query for an Outsourced Database

JINGJING GUO¹ AND JIACONG SUN²
¹State Key Laboratory of Integrated Service Networks, Xidian University, Xi’an 710071, China
²School of Electronic and Computer Engineering, Peking University Shenzhen Graduate School, Shenzhen 518055, China
Corresponding author: Jingjing Guo (jennifer.jing.sun@gmail.com)

ABSTRACT Cloud computing enables resource-constrained clients to outsource their computation-intensive data storage and computation tasks to a cloud server. Although it creates many benefits, data outsourcing causes some security challenges for clients. In this paper, we propose the secure and efficient query (SecNN) scheme to address the secure nearest neighbor query problem, which is popular in academia and industry. Compared with the state-of-the-art scheme, the proposed scheme can significantly decrease the number of communications between the clients and the cloud server. Furthermore, it achieves the desired security in the random oracle model. Moreover, the extending scheme described can verify the correctness and completeness of query results and database updates by using the Merkle hash tree technique. Finally, an experimental evaluation demonstrates the high efficiency of our scheme.

INDEX TERMS Cloud computing, outsourcing computation, database encryption, nearest neighbor query.

I. INTRODUCTION
Cloud computing has attracted considerable research interests in view of its powerful storage and computational capacity, which offers outsourcing services for resource-constrained clients [1]–[4]. Outsourcing enables clients to turn data into a valuable asset. It has been demonstrated as a potential tool and has numerous possible uses in big data [5]–[8]. However, the cloud server may be untrustworthy and eavesdrop on confidential information, such as financial records and personal health information.

One method for protecting data secrecy, which has been studied for many years, is to encrypt the outsourced database [9]–[12]. Unfortunately, traditional encryption schemes inevitably break the correlation among different data items, thus increasing the difficulty of some normal operations. For example, traditional encryption breaks the order information in plaintexts and makes order comparison on ciphertexts impossible. Although fully homomorphic encryption is a potential solution [9], the existing schemes are not practical for the sake of heavy computational time and ciphertext size.

The nearest neighbor query aims to identify the nearest data tuples in a database D based on a query point Q, which is one of the most basic operations in practice for processing potential applications in machine learning [13]–[15]. As a result, data secrecy in nearest neighbor queries becomes a crucial issue for the outsourced database. Recently, many researches have been conducted to protect data secrecy in nearest neighbor queries [16]–[18]. Based on asymmetric scalar-product-preserving encryption (ASPE), Wong et al. [16] proposed two schemes to support nearest neighbor query under different attack models. However, both schemes have a linear search complexity regarding the data items. Moving a step forward, Yao et al. [19] constructed a nearest neighbor query scheme based on partitions. Although the scheme achieves a faster-than-linear search complexity, it does not support the data items with a dimensionality greater than two. Subsequently, Wang et al. [17] proposed another scheme based on the tree structure and searchable encryption techniques. Their scheme adopted a lightweight cryptographic primitive, order-preserving encryption (OPE), to obtain a faster-than-linear search complexity. To the best of our knowledge, practical OPE schemes cannot simultaneously satisfy the requirements of noninteraction and indistinguishability under an ordered chosen-plaintext attack (IND-OCPA). Therefore, the state-of-the-art scheme [17] with IND-OCPA security needs \(O(\log n)\) rounds of interactions, where \(n\) is the number of data items in the database.

A. OUR CONTRIBUTIONS
In this paper, we focus on the construction of a secure and efficient nearest neighbor query scheme for the outsourced
database in cloud computing. The main contributions are as follows:

- We apply the prefix encoding and R-tree techniques to design a nearest neighbor query scheme (called SecNN) for the outsourced database. SecNN achieves index indistinguishability under the chosen keyword attack (IND-CKA) in the random oracle model.
- The proposed SecNN scheme only requires 2 rounds of interactions (same as plaintexts) between a client and a cloud server, while the state-of-the-art scheme [17] requires $O(\log n)$ rounds of interactions due to the OPE technique.
- We further extend SecNN against a malicious cloud server by incorporating the Merkle hash tree technique. Thus, clients can verify the correctness and completeness of the outsourced database and query results.

### B. RELATED WORK

The basis of the nearest neighbor query has been studied in the past decades [16], [17], [19]–[21]. In this section, we review some existing nearest neighbor query schemes and analyze their limitations in detail.

Wong et al. [16] proposed an ASPE algorithm, which preserved a special scalar product. They constructed two nearest neighbor query schemes by ASPE algorithm. Their schemes support the nearest neighbor (NN) computation and resist practical attacks of different background knowledge. Based on non-colluded cloud servers [22] and the Paillier cryptosystem [23], Elmehdwi et al. [24] designed a nearest neighbor query scheme. Yin et al. [25] introduced a privacy-preserving k-means clustering technology over encrypted multi-dimensional cloud data by leveraging the scalar-product-preserving encryption primitive. Unfortunately, all above schemes have a linear-search complexity regarding the database size. As an improvement, Hu et al. [20] designed a new nearest neighbor query scheme from privacy homomorphism (PH). PHs are encryption schemes that support computations on ciphertexts without decryption. The proposed scheme [20] supports large-scale databases and has a faster-than-linear search complexity by using the tree structure and ASPE. However, it requires a local index and $O(\log n)$ rounds of interactions between a client and a cloud server.

Recently, Yao et al. [19] firstly pointed out that the nearest neighbor problem was at least as hard as the construction of an OPE scheme, and showed that it was impossible to retrieve the NN given only $E(G)$ and $E(D)$ based on the impossibility of OPE in the standard model. For this reason, an algorithm based on the secure Voronoi diagram (SVD) technique was proposed to retrieve a partition $E(G)$, where $E(G)$ contains the NN. Unfortunately, it requires the client to locally maintain a large-size index. Wang et al. [17] designed a practical and secure nearest neighbor query scheme in cloud computing. They adopted a lightweight cryptography primitive OPE to construct a solution and achieved sublinear search complexity. OPE [26] was proposed to solve the range query problem. Subsequently, Boldyrev et al. [27] proved that IND-CPA security was unachievable for a practical OPE scheme unless it had an exponential ciphertext space. Furthermore, Popa et al. [28] pointed out that mutable ciphertext was required for IND-CPA security. To the best of our knowledge, all OPE schemes with IND-CPA security require $O(\log n)$ rounds of interactions. Therefore, Wang et al.’s scheme [17] requires $O(\log n)$ rounds of communication. Similarity, Hsu et al. [29] constructed a PPkNN nearest neighbor search scheme by leveraging the extension of OPE and R-tree. However, their scheme fails in the man-in-the-middle attack. Lei et al. [30] proposed a secure k-nearest neighbor (SkNN) query scheme from the projection function approach to code neighbor regions. This protocol only returns approximate results and donot support high-dimensional data.

Hacigümüs et al. [1] firstly proposed the idea of database outsourcing. Devanbu et al. [31] studied the verifiability of an outsourced database by using Merkle hash tree technique. However, their scheme did not consider the case in which a cloud server returns an empty result. Moreover, Wang et al. [32] proposed verifiable auditing for an outsourced database, especially when the cloud server returned an empty result. Cheng and Tan [33] verified that kNN results were complete, authentic and minimal. More promising approaches based on network Voronoi diagram (NVD) and neighbors have been invented. Hu et al. [34] studied the problem of privacy-preserving query authentication for location-based services of multidimensional indexes and later solved the problem of distance verification and path verification on road networks in [35].

### C. ORGANIZATION

The rest of this paper is organized as follows. We illustrate some preliminaries in section 2. The system model and the formal definition are shown in section 3. Furthermore, we describe the proposed SecNN scheme in the honest-but-curious cloud server model in section 4. In section 5, we further consider the verifiable update and query in the malicious cloud server model. Section 6 illustrates the security and efficiency analysis. The performance evaluation of SecNN scheme is presented in section 7. Finally, the conclusions are presented in section 8.

### II. PRELIMINARIES

We first provide a brief introduction to the definitions and properties of the Bloom filter, Merkle hash tree, and prefix encoding. Then we review the method to find the NN point based on an R-tree.

#### A. BLOOM FILTER

A Bloom filter refers to a space-efficient data storage structure. It is always used to determine whether an element belongs to a set with allowable errors [36]. A Bloom filter contains a binary array of size $m$ and $k$ independent hash functions $\{h_i : \{0, 1\}^* \rightarrow [1, m]|1 \leq i \leq k\}$. In the initial phase,
all positions of the array are set to 0. To insert an element \( x \), it computes \( k \) independent hash functions \( h_i(x)(1 \leq i \leq k) \) and modifies the corresponding positions of the array to 1, such as \( a_1, a_2 \). To query an element \( x \), it computes hash functions and determines whether the corresponding positions are 1. If there exists 0 at some positions, then \( x \) does not belong to the set \( S \), such as \( b_2 \). If values at the positions are equal to 1, then \( x \) belongs to set \( S \) with allowable errors, such as \( b_1, b_3 \).

In Fig. 1, it seems easy to draw a conclusion that \( b_2 \) does not belong to the set \( S \), while the Bloom filter determines that it belongs to set \( S \). In this condition, we call it false positive probability denoted \( P_f \). In [37], the author drew the conclusion \( P_f = (1 - e^{-kn/m})^k \), where \( n \) was the number of data items. The false positive \( P_f \) reaches the minimum value \((0.6185)^{m/n} \) if \( k = \ln 2 \times \frac{m}{n} \).

**B. MERKLE HASH TREE**

The Merkle hash tree is a special binary tree structure. It is used to authenticate the integrity of outsourced database [38]. A Merkle hash tree stores a hash of data items in leaf nodes and a hash of their children in non-leaf nodes. The clients only store the root locally and the cloud server stores the whole tree. In the verification phase, the cloud server returns the data and their sibling nodes to the root. Then, the client reconstructs the tree and determines whether the reconstructed root is equal to the stored hash. As shown in Fig. 2, to verify a set \( \{d_3, d_4\} \), the cloud server returns the dataset \( \{d_3, d_4\} \) and its sibling values \( h_{12}, h_{58} \). Afterward, the clients compute \( h'_{13} = h(d_3), h'_{14} = h(d_4) \) and \( h'_{34} = h(h'_{13} || h'_{14}) \). According to the sibling values, the clients reconstruct \( h'_{14} = h(h_{12} || h'_{34}) \) and \( h'_{18} = h(h'_{14} || h_{58}) \). Finally, they check if \( h'_{18} = h_{18} \). If it holds, the set is integrated and correct; otherwise, it has been tampered.

The Merkle hash tree has a lower computational and storage cost. The clients compute \( O(\log n) \) hashes which are very fast. Generally, the cloud server only stores data items in leaf nodes. When the clients want to verify a set, the cloud server temporarily generates the sibling values.

**C. PREFIX ENCODING**

The prefix encoding technique converts a range query problem to test whether two sets have common elements [39]. Given an element \( x \) of \( w \) bits \( x = b_1 b_2 \ldots b_w \), its prefix family \( F(x) \) is defined as \( w + 1 \) prefixes \( \{b_1 b_2 \ldots b_w, b_1 b_2 \ldots b_{w-1} *, \ldots, b_1 * * * \ldots * * \ldots * \} \). For a range \( [a, b] \), its range prefix \( S([a, b]) \) is defined as the minimum prefix set which covers the range \( [a, b] \). In this condition, we convert the testing of whether an element \( x \) falls into a range \( [a, b] \) to the testing of whether sets \( F(x) \) and \( S([a, b]) \) have common elements, due to \( x \in [a, b] \) iff \( F(x) \cap S([a, b]) \neq \emptyset \). An instance is shown as follows. Given 6 of 5 bits, its prefix family is \( F(5) = \{00110, 0011*, 001* *, 00* **, 0* ***, *, ** ***\} \), and the range prefix of \([0, 7]\) is \( S([0, 7]) = \{00 **\} \). Then, it is easy to see that \( 6 \in [0, 7] \) on account of \( F(6) \cap S([0, 7]) = \{00 **\} \).

**D. R-TREE**

An R-tree is a hierarchical data structure. It handles geometrical data, such as point, line, surface, and volume [40]. An R-tree always has a faster-than-linear search complexity and is generally efficient [41].

In an R-tree, geometric data are represented as a minimum bounding rectangle (MBR) and stored in a leaf node. The nonleaf node is an MBR that bounds its children. At the same time, an R-tree is a dynamic data structure that does not require global reorganization. We present the method for solving the nearest neighbor query problem with an R-tree in the following.

1) According to an NN query point \( Q \), a cloud server traverses the R-tree to find the deepest nonleaf node \( B_{deepest} \), which contains \( Q \) and returns the leaf nodes that exist under this non-leaf node \( B_{deepest} \).

2) The client computes the closest point \( d_i \) in \( B_{deepest} \) as a temporary NN and computes the smallest distance \( r \). Next, the client draws a circle \( O \) center at \( Q \) with radius \( r \) and generates a temporary rectangle \( R \) that is the MBR of circle \( O \).

3) The cloud server traverses the R-tree to determine whether some leaf nodes are in \( R \). The cloud server searches its children only if a nonleaf node \( B_i \) intersects with the temporary \( R \). When arriving at a leaf node, the cloud server determines whether the leaf node \( d_i \) falls into \( R \). If no points fall in \( R \), then the temporary NN
is the NN. Otherwise, the cloud server returns the leaf nodes. Afterward, the client computes the closest point \( d_k \) in \( R \) and takes it as the NN.

An instance is illustrated in Fig. 3. First, the cloud server finds the deepest nonleaf node \( B_4 \) that contains the query point \( Q \) and computes the temporary NN \( d_7 \). Then, the client computes its temporary rectangle \( R \) and submits it to the cloud server. Afterward, the cloud server finds that no leaf node falls in \( R \). Therefore, the temporary NN \( d_7 \) is the NN.

III. PROBLEM FORMULATION

A. SYSTEM MODEL

As shown in Fig. 4, the system model of our SecNN scheme includes two entities: a client and a cloud server. The client is an entity that outsources its encrypted database to save storage and computational burden and then provides a query request. The cloud server is an entity that stores the encrypted database, searches over it according to the query request and returns the corresponding result. We assume that the cloud server is untrusted and consider the following threat models [42].

- **Honest-but-Curious**: The cloud server follows a protocol honestly and returns the answer correctly to the client, but tries to learn information about the encrypted database. To protect its secrecy, the client encrypts the data items before outsourcing.

- **Malicious**: The cloud server is not only curious but also dishonest. It can misbehave in any way, such as storing damaged data or returning incorrect results. In this model, the client verifies the correctness and completeness of the outsourced database and query results.

B. FORMAL DEFINITION

A secure nearest neighbor query scheme empowers a client to outsource its database to a cloud server and then to provide the query request to search the NN for a given query point \( Q \). Wang et al. [17] provide a formal definition of the nearest neighbor query scheme.

**Definition 1**: A secure nearest neighbor query scheme is a tuple of polynomial-time algorithms \( \Pi = \{ \text{KeyGen}, \text{TreeBuild}, \text{Enc}, \text{TokenGen}, \text{Search} \} \) defined as follows:

- **KeyGen** (\( \lambda \)) \( \rightarrow \) \( sk \): Take a security parameter \( \lambda \) as input, this key generation algorithm outputs a secret key \( sk \).
- **TreeBuild** (\( D \)) \( \rightarrow \) \( \Gamma \): Take a database \( D = \{d_1, d_2, \ldots, d_n\} \) as input, this algorithm outputs an R-tree \( \Gamma = \{d_1, d_2, \ldots, d_n, B_1, B_2, \ldots, B_m, P\} \), where \( d_i \) is a leaf node that stores the data record, \( B_j \) is a nonleaf node and \( P \) is a set of pointers to cover the parent-child relations.
- **Enc** (\( sk, \Gamma \)) \( \rightarrow \) \( \Gamma^* \): Take a secret key and an R-tree \( \Gamma \) as inputs, this encryption algorithm outputs an encrypted R-tree \( \Gamma^* = \{\bar{d}_1, \bar{d}_2, \ldots, \bar{d}_n, \bar{B}_1, \bar{B}_2, \ldots, \bar{B}_m, \bar{P}\} \), where \( \bar{d}_i \) is an encrypted leaf node, \( \bar{B}_j \) is an encrypted nonleaf node and \( \bar{P} \) is a set of pointers to cover the parent-child relations in the encrypted R-tree \( \Gamma^* \).
- **TokenGen** (\( sk, Q \)) \( \rightarrow \) \( tk \): Take a secret key and a query point as inputs, this token generation algorithm outputs a token \( tk \) for the query point \( Q \).
- **Search** (\( tk, \Gamma^* \)) \( \rightarrow \) \( I \): Take a search token \( tk \) and an encrypted R-tree \( \Gamma^* \) as inputs, this search algorithm outputs the result \( I \).
In the following, we aim to introduce the IND-CKA security definition in [43]. Informally, it means that the adversary \( A \) cannot distinguish which index is built when the databases \( S_1 \) and \( S_2 \) have the same number of items.

Definition 2: Take a security parameter \( \lambda \) as input, a scheme \( \Pi \) is IND-CKA secure if for all probabilistic polynomial-time (PPT) adversaries \( A \),

\[
Adv^\Pi_A(k) \leq \text{negl}(k)
\]

where \( Adv^\Pi_A(k) = |Pr[b' = b] - \frac{1}{2}| \) is defined as the advantage of adversary \( A \) in the experiment as follows:

1. A challenger \( C \) generates a secret key \( sk \) ← \( \text{KeyGen}(\lambda) \).
2. An adversary \( A \) chooses two different databases \( S_0 \) and \( S_1 \), where both databases have the same number of data items. Then, adversary \( A \) gives \( S_0 \) and \( S_1 \) to the challenger \( C \).
3. After receiving databases \( S_0 \) and \( S_1 \), the challenger \( C \) chooses \( b \) ← \( \{0, 1\} \), constructs index \( I \) for database \( S_b \), and gives it to the adversary \( A \).
4. The adversary \( A \) outputs \( b' \), its guess for \( b \).

IV. THE PROPOSED SECNN SCHEME

A. HIGH DESCRIPTION

In this paper, we propose a secure SecNN scheme based on an R-tree to solve the privacy-preserving problem in the nearest neighbor query. In the proposed scheme, the cloud server needs to test whether a query point \( Q \) falls into an MBR \( B_1 \), whether a leaf node \( d_i \) falls into a temporary rectangle \( R \), and whether an MBR \( B_1 \) intersects with \( R \).

Fortunately, we can take the Bloom filter and prefix encoding techniques to address all the above issues. For an MBR \( B_1 = (x_{i1}, x_{i2}) \times (y_{i1}, y_{i2}) \) and data record \( d_i = (x_i, y_i) \), we encode them as \( [B_1] = (F(x_{i1}), F(x_{i2})) \times (F(y_{i1}), F(y_{i2})) \) and \( [d_i] = (F(x_i), F(y_i)) \). For a temporary rectangle \( R = (x_{r1}, x_{r2}) \times (y_{r1}, y_{r2}) \) and query point \( Q = (x, y) \), we encode them as \( [R] = (S([0, x_{r1}]), S([0, x_{r2}])) \times (S([0, y_{r1}]), S([0, y_{r2}])) \) and \( [Q] = (S([0, x]), S([0, y])) \). After encoding, the above problems convert to test whether two sets have common elements. We take the testing of whether a query point \( Q \) falls into an MBR \( B_1 \) as an example. The condition in which \( Q \) falls in an MBR \( B_1 \) is equivalent to the conditions \( x_{i1} < x < x_{i2}, y_{i1} < y < y_{i2} \). Moreover, we know that \( \alpha \in [c, d] \) iff \( F(\alpha) \cap S([c, d]) \neq \emptyset \). Therefore, the problem converts to testing \( S([0, x]) \cap F(x_{i1}) \neq \emptyset \), \( S([0, x]) \cap F(x_{i2}) \neq \emptyset \), \( S([0, y]) \cap F(y_{i1}) \neq \emptyset \), \( S([0, y]) \cap F(y_{i2}) \neq \emptyset \). Finally, the cloud server searches the encrypted database and returns the result.

B. MAIN CONSTRUCTION

The proposed SecNN scheme \( \Pi = \{\text{KeyGen, TreeBuild, Enc, TokenGen, Search}\} \) consists of five algorithms defined as follows:

- **KeyGen(\( \lambda \))**: On input a security parameter \( \lambda \), the client chooses an IND-CPA secure symmetric encryption scheme \( \text{DET} = (\text{Gen, Enc, Dec}) \) and a hash function \( H \) with secret keys \( sk_2 = (k_1, k_2, \ldots, k_l) \), which are used to construct a Bloom filter \( BF \). The client has secret keys \( sk = (sk_1, sk_2) = (k, k_1, k_2, \ldots, k_l) \), where \( sk_1 = k \) ← \( \text{Gen}(\lambda) \).
- **TreeBuild(\( D \))**: On input a database \( D \), the client builds an R-tree \( \Gamma = \{d_1, d_2, \ldots, d_n, B_1, B_2, \ldots, B_m, P\} \).
- **Enc(\( sk, \Gamma \))**: Given a secret key \( sk \) and R-tree \( \Gamma \), the client proceeds to perform data encryption and index preserving. The details of both algorithms are described as follows.

1. **Data Encryption**. This algorithm encrypts a leaf node (i.e. data record) \( d_i = (x_i, y_i) \) as
   \[
   \tilde{d}_i = (\tilde{x}_i, \tilde{y}_i), \quad 1 \leq i \leq n,
   \]
   where \( \tilde{x}_i = \text{DET.Enc}(k, x_i), \tilde{y}_i = \text{DET.Enc}(k, y_i) \).
2. **Index Preserving**. To facilitate the data search, the client constructs an index. Furthermore, he encrypts the index into ciphertexts to protect data secrecy. The specific construction is as follows.
   a) Index Construction. For a leaf node \( d_i = (x_i, y_i), (1 \leq i \leq n) \) and a nonleaf node MBR \( B_j = (x_{j1}, x_{j2}) \times (y_{j1}, y_{j2}), (1 \leq j \leq m) \), the client encodes them as prefix families
   \[
   [d_i] = (F(x_i), F(y_i)),
   [B_j] = (F(x_{j1}), F(x_{j2})) \times (F(y_{j1}), F(y_{j2})).
   \]
   Furthermore, the client computes their Bloom filters as
   \[
   BF_{d_i} = (BF_{x_i}, BF_{y_i}),
   BF_{B_j} = (BF_{x_{j1}}, BF_{x_{j2}}, BF_{y_{j1}}, BF_{y_{j2}}).
   \]
   b) Node Randomization. For each node, we generate a random number \( v.R \), namely, we compute \( H(v.R, H(k_{1}), P), H(v.R, H(k_{2}), P), \ldots, H(v.R, H(k_{r}), P) \) to obtain their Bloom filter values for every prefix \( P \), as shown in Fig. 5.

Then the client updates the encrypted database and index tree to the cloud server, where \( \Gamma^* = \{\tilde{d}_1, \ldots, \tilde{d}_n\} \).
Later, the client computes its search token as a hash $M$.

Then, the client submits the search token $M$ into the temporary rectangle.

### TokenGen($sk, Q$)

For a query point $Q = (x, y)$, the client firstly computes its range prefix as

$$[Q] = (S([0, x_1]), S([0, y_1]))$$

Then, the client computes its search token as a hash matrix $M_Q$

$$M_Q = \{M_x, M_y\}$$

Later, the client computes its search token as a hash matrix $M_Q$

$$M_Q = \{M_x, M_y\}$$

Then, the client submits the search token $M_Q$ to the cloud server.

### Search($M_Q, \Gamma^*$)

Given a search token $M_Q$ and an encrypted tree $\Gamma^*$, the cloud server computes the deepest nonleaf node $B_1$ and returns the data records in $B_1$ to the client. After receiving the encrypted data records in $B_1$, the client decrypts them and computes the temporary closest point $T$ as a temporary NN. Next, it generates a temporary rectangle $R = (x_{r1}, x_{r2}) \times (y_{r1}, y_{r2})$ and gives it to the cloud server. Then the cloud server searches the encrypted tree $\Gamma^*$ to test whether some leaf nodes fall into the temporary rectangle $R$.

In the Search phase, the cloud server proceeds three operations: whether a query point $Q$ is in an MBR $B_1$, whether a temporary rectangle $R$ intersects with an MBR $B_1$, and whether a leaf node $d_i$ is in a temporary rectangle. We solve all these problems in the following:

1) Whether a query point $Q$ is in an MBR $B_1$. In this step, the cloud server owns $M_Q = \{M_x, M_y\}$ and $BF_B$. Owing to the fact that $x \in [a, b]$ if $F(x) \cap S([a, b]) \neq \phi$, the cloud server verifies the relations $x_{r1} < x < x_{r2}, y_{r1} < y < y_{r2}$ in plaintexts. Namely, it verifies the following relations in ciphertexts:

$$F(x_{r1}) \cap S([0, x_1]) \neq \phi, F(x_{r2}) \cap S([0, x_1]) = \phi;$$

$$F(y_{r1}) \cap S([0, y_1]) \neq \phi, F(y_{r2}) \cap S([0, y_1]) = \phi.$$

The inequation $F(x_{r1}) \cap S([0, x_1]) \neq \phi$ means that there exists a row $i$ such that $BF_{y_1}(M_x) = 1$. On the contrary, the equation $F(x_{r2}) \cap S([0, x_1]) = \phi$ means that no row in $M_x$ satisfies $BF_{x_2}(M_x) = 1$.

According to the search token $M_Q$ and MBR $BF_B$, the cloud server finds the deepest nonleaf node $B_{\text{deepest}}$ and returns it to the client.

2) Whether a temporary rectangle $R$ intersects with an MBR $B_1$. For a temporary rectangle $R = (x_{r1}, x_{r2}) \times (y_{r1}, y_{r2})$, its token is generated as $M_R = (M_{x_{r1}}, M_{x_{r2}}, M_{y_{r1}}, M_{y_{r2}})$. From the root to a leaf node, the cloud server tests whether a nonleaf node $B_1$ intersects with $R$. If intersecting, the cloud server continues to search the children. Otherwise, the searching progress terminates in this branch. We present it in Algorithm 1.

### Algorithm 1 Intersection Test

**Require:**

Given a rectangle token $R = (M_{x_{r1}}, M_{x_{r2}}, M_{y_{r1}}, M_{y_{r2}})$ and an index $B = (BF_{x_1}, BF_{x_2}, BF_{y_1}, BF_{y_2})$.

**Ensure:**

Output true if $B \cap R = \phi$; otherwise output false.

1: RESULT $= true$;
2: If $BF_{x_1}(M_{x_{r2}}) = 0$ or $BF_{x_2}(M_{x_{r1}}) = 1$ or $BF_{y_1}(M_{y_{r2}}) = 0$
3: then RESULT $= false$;
4: return RESULT.

3) Whether a leaf node $d_i$ is in a temporary rectangle $R$. Arriving at the leaf nodes, the cloud server retrieves a leaf node which is in the temporary rectangle $R$. The algorithm is shown in Algorithm 2. At the same time, the results may have some redundancies, namely, many elements may fall in the temporary rectangle $R$. If the redundancy is too large, the client randomly chooses a leaf node as the temporary NN to carry out the Search algorithm.

### Algorithm 2 Inside Test

**Require:**

Given a rectangle token $R = (M_{x_{r1}}, M_{x_{r2}}, M_{y_{r1}}, M_{y_{r2}})$ and a leaf node $d_i = (BF_{x_1}, BF_{y_1})$.

**Ensure:**

Output true if $d \in R$; otherwise output false.

1: RESULT $= false$;
2: If $BF_{x_1}(M_{x_{r1}}) = 0$ and $BF_{x_2}(M_{x_{r2}}) = 1$ and $BF_{y_1}(M_{y_{r2}}) = 0$
3: then RESULT $= true$;
4: return RESULT.

### V. EXTENDING SCHEME: A VERIFIABLE SECNN SCHEME

A malicious cloud server is a cloud server that may perform operations dishonestly and return false results. For example, it may not insert a new value into the database or may return an empty result to save storage and computation cost. To avoid these false operations, the client adds Merkle hashes [44] to verify the correctness and completeness of the outsourced database and the query results.

As indicated in Fig. 6, we present the result of adding a Merkle hash on an R-tree. The hash is computed over its Bloom filters and Merkle hashes of its children. The client memorizes the root to verify whether an operation was performed correctly by the cloud server. For example, when
verifying a data record $a$ is in the database, the cloud server provides a sibling-path to the root. If the data record $a$ is in the database, the client reconstructs a root Merkle hash and determines if it equals the stored hash. Otherwise, there exists a level $i$ such that $BF(i,j)(a) = 0$ for all $j$. $BF(i,j)(a) = 0$ means that the original $BF(i+1,j−1)(a) = 0$ and $BF(i+1,j)(a) = 0$. In this condition, the client reconstructs a root and verifies it. The completeness of a database can be verified as above. For an NN query, the client only needs to verify its correctness. In the following, we present a verification method in detail.

A. PROOFS OF INSERTION AND DELETION
When a client performs the insertion or deletion operation, the cloud server needs to provide a proof that the database has been updated correctly. The proof is presented as follows:

- **Old Merkle Information.** The information was affected by the insertion or deletion and the information came from their sibling nodes.
- **New Merkle Information.** The information forms new sibling nodes after insertion or deletion.

After receiving the proofs, the client checks as follows. He first verifies that the insertion or deletion was executed correctly by computing the old root based on the old Merkle information. The client tests if it agrees with the stored root. If it agrees, the update operation has been executed correctly. Whereafter, the client regenerates the root and stores the new root of the Merkle hash tree.

B. PROOFS OF SEARCH
To prove the correctness, the client regenerates the root hash based on the results and their sibling nodes. Then the client verifies if it agrees with the stored root. In the nearest neighbor query algorithm, the cloud server verifies that the result is the NN. In this step, the client regenerates a rectangle $R'$ based on the result and verifies that no leaf nodes fall into $R'$. This process is similar to the verification of whether a data record $a$ is in the database.

VI. PERFORMANCE ANALYSIS
A. SECURITY ANALYSIS
In this section, we prove that the proposed scheme SecNN achieves IND-CKA security.

**Theorem 1:** The proposed nearest neighbor query scheme is IND-CKA secure in the random oracle model.

**Proof:** Let $f : \{0, 1\}^n \times \{0, 1\}^s \rightarrow \{0, 1\}^m$ be a keyed hash function, which takes $n$-bit strings and $s$-bit key strings as input, and outputs $m$-bit strings. Let $RF_n : \{0, 1\}^{2n} \rightarrow \{0, 1\}^m$ be a random function family. A function $g$ is pseudorandom if it satisfies computational and pseudorandom. $g(x, k)$ can be computed by a PPT algorithm that is given input $x$ and $k$. Therefore, for all PPT adversaries $A$, $\{k \rightarrow \{0, 1\}; g(., k)\} \approx \{F \leftarrow_{R} RF_n; F\}$.

We prove the proposed scheme by contradiction. We suppose that the proposed scheme is not IND-CKA secure, namely, there exists an adversary $A$ which can distinguish the index with non-negligible probability. After, we construct a PPT adversary $B$ using $A$ as a subroutine to distinguish a pseudorandom function from a random function. If $A$ can break with non-negligible probability, and thus adversary $B$ can distinguish with non-negligible probability. The algorithm works as follows.

Given a function $f$, $f$ is a pseudorandom function or random function, a challenger $C$ replaces hash $h$ with $f$. Note that, $C$ can compute function $f$ through to access its oracle $O_f$. Algorithm $B$ takes algorithm $A$ as its subroutine in the following game.

Takes an unknown function $f$ as input, algorithm $B$ wants to decide whether it is a pseudorandom function or random function, $B$ calls algorithm $A$ to choose sets $S_0$ and $S_1$, where $S_0$ and $S_1$ have the same number of data items. $B$ sends sets $S_0$ and $S_1$ to the challenger $C$. After receiving the sets, $C$ randomly chooses $S_0, b \in R \{0, 1\}$ to construct its index as $I_b$ and sends it to an adversary $A$. $A$ calls $f$ to guess which set that $I_b$ is built for. $A$ outputs $b'$ as a guess for $b$. If $b' = b$, then $A$ outputs 1, indicating $f$ is a pseudorandom function. Otherwise, $A$ outputs 0.

In the following, we show that if $A$ can break with non-negligible probability, then algorithm $B$ can distinguish a pseudorandom function from a random function with non-negligible probability by proving the following claims.

- **Claim 1:** When $f$ is a pseudorandom function, then

$$|Pr[f^{(\cdot,k)} = 1 | k \rightarrow [0, 1]^s - \frac{1}{2}]| \geq \text{negl}(\lambda).$$

- **Claim 2:** When $f$ is a random function, then

$$Pr[B^{(\cdot)} f \leftarrow RF_n] = \frac{1}{2}.$$
TABLE 1. Comparison among existing schemes and SecNN scheme.

| Scheme           | Interaction | Faster-than-Linear | Single Server |
|------------------|-------------|--------------------|---------------|
| Wong et al. [16] | $O(\log n)$ | $\times$           | $\sqrt{\ }$  |
| Wang et al. [17] | $O(\log n)$ | $\sqrt{\ }$       | $\sqrt{\ }$  |
| Elmeleeh et al. [24] | $1$ | $\times$ | $2$ |
| SecNN scheme     | $2$         | $\sqrt{\ }$       | $\sqrt{\ }$  |

(if there are more than one Bloom filters in each node, we will assign different random numbers for different Bloom filters), and function $f$ uses random number $v$. 

Facing a random position of Bloom filter $B_v$, adversary $A$ guesses index $b$ what is built for. Therefore, adversary $A$ guesses $b$ correctly with probability of $\frac{1}{2}$. It follows that $\Pr[B_f = 1 | f \leftarrow RF_n] = \frac{1}{2}$.

From Claim 1 and Claim 2, we can draw the conclusion that $B$ can distinguish a pseudorandom function from a random function, which is impossible. Thus we prove that the proposed scheme is IND-CKA secure.

B. COMPLEXITY ANALYSIS

Compared to the other nearest neighbor query schemes, our SecNN scheme shows advantages in communication and search efficiency. The details are shown in Table 1. Furthermore, we present the efficiency analysis between the state-of-the-art scheme [17] and SecNN scheme.

For the convenience of description, some marks are introduced. We denote by $E/D$ an encryption/decryption of an IND-CPA secure symmetric encryption scheme, by $H$ a hash, by $m$ the number of MBR $B_i$ in an R-tree, by $n$ the size of database $D$, by $d$ the dimensions of data in $D$, by $w$ the bits of data, by $l_1$ the number of data in the deepest nonleaf node $B_{\text{deepest}}$, and by $l_2$ the number of data in the temporary rectangle $R$.

In Wang et al.’s scheme [17], they used OPE to test the order information. As demonstrated in the scheme [28], we can draw the conclusion that the encryption/decryption computation of OPE equals to the rounds of communication multiplied by the decryption computation of DET, namely, $E_{\text{OPE}}/D_{\text{OPE}} = \log n \times D$. In SecNN scheme, we use prefix encoding and Bloom filter techniques to replace OPE. It focuses on the hash computation and we present it in Table 2.

VII. PERFORMANCE EVALUATION

In this section, we present a formal experimental evaluation of the SecNN scheme. More specifically, we run our SecNN with the C++ language on a machine with an Intel Xeon(R) CPU E5-2609 v3 processor running at 1.90 GHz and 96 GB memory. The cipher used is AES-ECB-256, as provided by the Crypto library. Throughout the experiment, we simulate both the cloud server and the client on the same machine.

A. DATABASE

In the experiment, we choose 1,000,000 data items randomly from the Gowalla database, which has 6,442,890 check-in locations collected from 196,591 users.

B. NETWORK

We note that scheme [17] will be affected by the communication due to the existence of $O(\log n)$ interactions, where $n$ is the size of data items in a database. The communication latency differs with the variance of network environment. In this paper, we assume that the network is slower than 5 ms of latency and 20 Mbps bandwidth [45]. As shown in Table 3, it can be seen that the communication time of scheme [17] rapidly increases with the database. There are two rounds of communication in SecNN and approximately $O(\log n)$ rounds of communication in Wang et al.’s scheme.

Because the latency affects the running time of both schemes, we consider two network environments: (1) a network with communication latency and (2) a network without latency.

We provide the storage and time cost of the tree construction in Fig. 7. In our experiment, the tree construction consists of two factors: $\text{TreeBuild}$ and $\text{Enc}$. It is easy to see

### Table 2. Computation cost between scheme [17] and SecNN.

| Schemes          | Wang et al. [17] | SecNN scheme |
|------------------|------------------|--------------|
| $\text{Enc}$     | $nd \cdot E + (nd + 2md^2) \log n \cdot D$ | $nd \cdot E + (nd + 2md^2)(w + 1)l \cdot H$ |
| $\text{TokenGen}$ | $d \log n \cdot D$ | $d(2w - 2)l \cdot H$ |
| $\text{Search(First Round)}$ | $(l_1 d + 2d^2 \log^2 n) \cdot D$ | $l_1 d \cdot D + 2d^2 \log n \cdot (2w - 2)l \cdot H$ |
| $\text{Search(Second Round)}$ | $(l_2 d + 2d^2 \log n(1 + \log n)) \cdot D$ | $l_2 d \cdot D + 2d^2(1 + \log n) \cdot (2w - 2)l \cdot H$ |

### Table 3. Communication latency.

| Data item ($10^4$) | 10  | 20  | 30  | 40  | 50  | 60  | 70  | 80  | 90  | 100 |
|--------------------|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| Time cost (ms)     |     |     |     |     |     |     |     |     |     |     |
| SecNN              | 10  | 10  | 10  | 10  | 10  | 10  | 10  | 10  | 10  | 10  |
| Scheme [17]        | 98  | 98  | 101 | 103 | 104.5 | 106 | 107 | 108 | 109 | 109.5 |

VOLUME 8, 2020 83761
that the tree construction time and storage increase with the number of data items. The storage cost of both schemes has a linear-increase and the proposed SecNN scheme is higher than scheme [17]. The time cost of tree construction has a similar trend to the storage cost. However, the tree construction is only a one-time cost, and the cost of SecNN scheme
is efficient. Therefore, the proposed scheme is suitable for real-world applications.

As shown in Figs. 8 and 9, we provide the efficiency comparison of the nearest neighbor query with the increasing number of data items. In our experiment, we divide the Search phase in two rounds and simulate the experiments into two scenarios. In Figs. 8a and 9a, we simulate the cloud server and the client on the same machine without considering the communication latency. From the figures, we can conclude that both schemes achieve an average query processing time of about 10 milliseconds on the 1 million data items and are very efficient. In Figs. 8b and 9b, we present the impact of communication latency for both SecNN and Scheme [17]. These figures demonstrate that the communication latency sharply influences the search time in the scheme [17]. We test the client-side computation cost in both schemes. The client computation is about 12 ms in Wang et al. scheme [17] and 14 ms in our SecNN scheme. The SecNN scheme is slightly higher but acceptable. For these experiments, we did not actually transfer the data over a network, but merely measured the theoretical communication cost. If considering the response times of the client and the cloud server, the performance has a 10x slowdown. From the Figs. 8 and 9, it is easy to conclude that the proposed SecNN scheme is efficient in practice.

VIII. CONCLUSION

The nearest neighbor query is a normal query operation in many engineering disciplines. In this paper, we introduce a SecNN scheme which enables privacy-preserving nearest neighbor query in cloud computing. The key novelty of the proposed scheme is in leveraging prefix encoding technique to convert range query to test whether two sets have common elements. To achieve a practical performance, SecNN applies R-tree on a distributed database to facilitate the query. Additionally, it supports dynamic database and is secure in the random oracle model. Finally, the proposed scheme is implemented on the Gowalla database. It is shown that SecNN scheme can achieve high-efficiency.

ACKNOWLEDGMENT

The authors are grateful to the anonymous referees for their invaluable suggestions.

REFERENCES

[1] H. Hacigumus, B. Iyer, and S. Mebrota, “Providing database as a service,” in Proc. 18th Int. Conf. Data Eng., San Jose, CA, USA, 2002, pp. 29–38.
[2] X. Chen, J. Li, J. Ma, Q. Tang, and W. Lou, “New algorithms for secure outsourcing of modular exponentiations,” IEEE Trans. Parallel Distrib. Syst., vol. 25, no. 9, pp. 2386–2396, Sep. 2014.
[3] H. Huang, X. Chen, Q. Wu, X. Huang, and J. Shen, “Bitcoin-based fair payments for outsourcing computations of fog devices,” Future Gener. Comput. Syst., vol. 78, pp. 850–858, Jan. 2018.
[4] Z. Zhang, X. Chen, J. Li, X. Tao, and J. Ma, “HVDB: A hierarchical verifiable database scheme with scalable updates,” J. Ambient Intell. Humanized Comput., vol. 10, no. 8, pp. 3045–3057, Aug. 2019.
[5] M. J. Atallah, K. N. Pantazopoulos, J. R. Rice, and E. E. Spafford, “Secure outsourcing of scientific computations,” Adv. Comput., vol. 54, pp. 215–272, Jul. 2001.
[6] X. Chen, J. Li, X. Huang, J. Ma, and W. Lou, “New publicly verifiable databases with efficient updates,” IEEE Trans. Dependable Secure Comput., vol. 12, no. 5, pp. 546–556, Sep. 2015.
[7] J. Wang, X. Chen, J. Li, J. Zhao, and J. Shen, “Towards achieving flexible and verifiable search for outsourced database in cloud computing,” Future Gener. Comput. Syst., vol. 67, pp. 266–275, Feb. 2017.
[8] X. Zhang, T. Jiang, K.-C. Li, A. Castiglione, and X. Chen, “New publicly verifiable computation for batch matrix multiplication,” Inf. Sci., vol. 479, pp. 664–678, Apr. 2019.
[9] C. Gentry, “Fully homomorphic encryption using ideal lattices,” in Proc. 41st Annu. ACM Symp. Theory Comput. (STOC). Rockville, MD, USA: Bethesda Softworks, 2009, pp. 169–178.
[10] M. Armbrust, A. Fox, and R. Griffith, “A view of cloud computing,” Commun. ACM, vol. 53, no. 4, pp. 50–58, Apr. 2010.
[11] X. Chen, X. Huang, J. Li, J. Ma, W. Lou, and D. S. Wong, “New algorithms for secure outsourcing of large-scale systems of linear equations,” IEEE Trans. Inf. Forensics Security, vol. 10, no. 1, pp. 69–78, Jan. 2015.
[12] X. Chen, J. Li, J. Wang, J. Ma, and W. Lou, “Verifiable computation over large database with incremental updates,” IEEE Trans. Comput., vol. 65, no. 10, pp. 3184–3195, Oct. 2016.
[13] R. Bost, R. A. Popa, S. Tu, and S. Goldwasser, “Machine learning classification over encrypted data,” in Proc. New. Distrib. Syst. Secure. Symp., San Diego, CA, USA, 2015, p. 4325.
[14] X. Ma, F. Zhang, X. Chen, and J. Shen, “Privacy preserving multi-party computation delegation for deep learning in cloud computing,” Inf. Sci., vol. 459, pp. 103–116, Aug. 2018.
[15] X. Zhang, X. Chen, J. Wang, Z. Zhan, and J. Li, “Verifiable privacy-preserving single-layer perceptron training scheme in cloud computing,” Soft. Comput., vol. 22, no. 23, pp. 7119–7132, Dec. 2018.
[16] W. K. Wong, D. W.-L. Cheung, B. Kao, and N. Mamoulis, “Secure kNN computation on encrypted databases,” in Proc. 35th SIGMOD Int. Conf. Manage. Data (SIGMOD), Providence, RI, USA, 2009, pp. 139–152.
[17] B. Wang, Y. Hou, and M. Li, “Practical and secure nearest neighbor search on encrypted large-scale data,” in Proc. 35th Annu. IEEE Int. Conf. Comput. Commun. (INFOCOM), San Francisco, CA, USA, Apr. 2016, pp. 1–9.
[18] Y. Zhu, Y. Zhang, X. Li, H. Yan, and J. Li, “Improved collision-resisting secure nearest neighbor query over encrypted data in cloud,” Concurrency Comput., Pract. Exper., vol. 31, no. 21, Nov. 2019, Art. no. e4681.
[19] B. Yao, F. Li, and X. Xiao, “Secure nearest neighbor revisited,” in Proc. 29th Int. Conf. Data Eng. (ICDE), Brisbane, QLD, Australia, Apr. 2013, pp. 733–744.
[20] H. Hsu, J. Xu, C. Chi, and B. Choi, “Processing private queries over untrusted data cloud through privacy homomorphism,” in Proc. IEEE 27th Int. Conf. Data Eng., Hannover, Germany, Apr. 2011, pp. 601–612.
[21] Y. Zhu, Z. Huang, and T. Takagi, “Secure and controllable k-NN query over encrypted data cloud with key confidentiality,” J. Parallel Distrib. Comput., vol. 89, pp. 1–12, Mar. 2016.
[22] S. Bugiel, S. Nurnberger, A. Sadeghi, and T. Schneider, “Twin clouds: An architecture for secure cloud computing,” in Proc. Workshop Cryptogr. Secure Clouds (WCS), Zürich, Switzerland, 2011, pp. 32–44.
[23] P. Paillier, “Public-key cryptosystems based on composite degree residuosity classes,” in Proc. Int. Conf. Theory Appl. Cryptograph. Techn. (EUROCRYPT), Prague, Czech Republic, 1999, pp. 223–238.
[24] Y. Elmelwdi, B. K. Samantha, and W. Jiang, “Secure k-nearest neighbor query over encrypted data in outsourced environments,” in Proc. IEEE 30th Int. Conf. Data Eng., Chicago, IL, USA, Mar. 2014, pp. 664–675.
[25] H. Yin, J. Zhang, Y. Xiong, X. Huang, and T. Deng, “PPK-means: Achieving privacy-preserving clustering over encrypted multi-dimensional cloud data,” Electronics, vol. 7, no. 11, pp. 310–327, 2018.
[26] R. Agrawal, J. Kiernan, R. Sikand, and Y. Xu, “Order preserving encryption for numeric data,” in Proc. ACM SIGMOD Int. Conf. Manage. Data (SIGMOD), Paris, France, 2004, pp. 563–574.
[27] A. Boldyreva, N. Chenette, Y. Lee, and A. O’Neill, “Order-preserving symmetric encryption,” in Proc. Int. Conf. Theory Appl. Cryptograph. Techn. (EUROCRYPT), Cologne, Germany, 2009, pp. 224–241.
[28] R. A. Popa, F. H. Li, and N. Zeldovich, “An ideal-security protocol for order-preserving encoding,” in Proc. IEEE Symp. Secur. Privacy, Berkeley, CA, USA, May 2013, pp. 463–477.
[29] Y.-C. Hsu, C.-H. Hsueh, and J.-L. Wu, “A privacy preserving cloud-based k-NN search scheme with lightweight user loads,” Computers, vol. 9, no. 1, pp. 1–26, 2020.
[30] 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 Proc. IEEE 35th Int. Conf. Data Eng. (ICDE), Macau, China, Apr. 2019, pp. 662–673.

[31] P. Devanbu, M. Gertz, C. Martel, and S. G. Stubblebine, “Authentic third-party data publication,” in Proc. IFIP Annu. Work. Conf. Database Secur., Amsterdam, The Netherlands, 2000, pp. 101–112.

[32] J. Wang, X. Chen, X. Huang, I. You, and Y. Xiang, “Verifiable auditing for outsourced database in cloud computing,” IEEE Trans. Comput., vol. 64, no. 11, pp. 3293–3303, Nov. 2015.

[33] W. Cheng and K.-L. Tan, “Query assurance verification for outsourced multi-dimensional databases,” J. Comput. Secur., vol. 17, no. 1, pp. 101–126, Mar. 2009.

[34] H. Hu, J. Xu, Q. Chen, and Z. Yang, “Authenticating location-based services without compromising location privacy,” in Proc. Int. Conf. Manage. Data (SIGMOD), Scottsdale, AZ, USA: ACM, 2012, pp. 301–312.

[35] Y. Jing, L. Hu, W.-S. Ku, and C. Shahabi, “Authentication of k nearest neighbor query on road networks,” IEEE Trans. Knowl. Data Eng., vol. 26, no. 6, pp. 1494–1506, Jun. 2014.

[36] B. H. Bloom, “Space/time trade-offs in hash coding with allowable errors,” Commun. ACM, vol. 13, no. 7, pp. 422–426, Jul. 1970.

[37] M. Mitzenmacher, “Compressed Bloom filters,” IEEE/ACM Trans. Netw., vol. 10, no. 5, pp. 604–612, Oct. 2002.

[38] R. C. Merkle, “Protocols for public key cryptosystems,” in Proc. IEEE Symp. Secur. Privacy, Oakland, CA, USA, Apr. 1980, pp. 122–134.

[39] P. Gupta and N. McKeown, “Dynamic algorithms with worst-case performance for packet classification,” in Proc. Int. Conf. Broadband Commun., High Perform. Netw., Perform. Commun. Netw. (NETWORKING), Paris, France, 2000, pp. 528–539.

[40] M. Hadjieleftheriou, Y. Manolopoulos, Y. Theodoridis, and V. J. Tsotras, “R-trees: A dynamic index structure for spatial searching,” in Proc. ACM Int. Conf. Manage. Data (SIGMOD), Boston, MA, USA, 1984, pp. 47–57.

[41] Y. Manolopoulos, A. Nanopoulos, A. N. Papadopoulos, and Y. Theodoridis, R-Trees: Theory and Applications (Advanced Information and Knowledge Processing). London, U.K.: Springer, 2006.

[42] O. Goldreich, S. Micali, and A. Wigderson, “How to play any mental game, or a completeness theorem for protocols with honest majority,” in Proc. Annu. ACM Symp. Theory Comput. (STOC), New York, NY, USA, 1987, pp. 218–229.

[43] E. Goh, “Secure indexes,” Stanford Univ., Stanford, CA, USA, Tech. Rep., 2004.

[44] R. C. Merkle, “A certified digital signature,” in Proc. Int. Conf. Cryptol. (CRYPTO), Santa Barbara, CA, USA, 1989, pp. 218–238.

[45] D. S. Roche, D. Apon, S. G. Choi, and A. Yerukhimovich, “POPE: Partial order preserving encoding,” in Proc. ACM SIGSAC Conf. Comput. Commun. Secur. (CCS), Vienna, Austria, 2016, pp. 1131–1142.