Secure $k$-Nearest Neighbor Query over Encrypted Data in Outsourced Environments

Yousef Elmehdwi, Bharath K. Samanthula, and Wei Jiang

Department of Computer Science, Missouri S&T
500 West 15th Street, Rolla, MO 65409, USA
Email: {ymez76, bspq8, wjiang}@mst.edu

Abstract—For the past decade, query processing on relational data has been studied extensively, and many theoretical and practical solutions to query processing have been proposed under various scenarios. With the recent popularity of cloud computing, users now have the opportunity to outsource their data as well as the data management tasks to the cloud. However, due to the rise of various privacy issues, sensitive data (e.g., medical records) need to be encrypted before outsourcing to the cloud. In addition, query processing tasks should be handled by the cloud; otherwise, there would be no point to outsource the data at the first place. To process queries over encrypted data without the cloud ever decrypting the data is a very challenging task. In this paper, we focus on solving the $k$-nearest neighbor ($k$NN) query problem over encrypted database outsourced to a cloud: a user issues an encrypted query record to the cloud, and the cloud returns the $k$ closest records to the user. We first present a basic scheme and demonstrate that such a naive solution is not secure. To provide better security, we propose a secure $k$NN protocol that protects the confidentiality of the data, the user’s input query, and data access patterns. Also, we empirically analyze the efficiency of our protocols through various experiments. These results indicate that our secure protocol is very efficient on the user end, and this lightweight scheme allows a user to use any mobile device to perform the $k$NN query.

I. INTRODUCTION

As an emerging computing paradigm, cloud computing attracts many organizations to consider utilizing the benefits of a cloud in terms of cost-efficiency, flexibility, and offload of administrative overhead. In cloud computing model [1], [2], a data owner outsources his/her database $T$ and the DBMS functionalities to the cloud that has the infrastructure to host outsourced databases and provides access mechanisms for querying and managing the hosted database. On one hand, by outsourcing, the data owner gets the benefit of reducing the data management costs and improves the quality of service. On the other hand, hosting and query processing of data out of the data owner control raises security challenges such as preserving data confidentiality and query privacy.

One straightforward way to protect the confidentiality of the outsourced data from the cloud as well as from the unauthorized users is to encrypt data by the data owner before outsourcing [3]. By this way, the data owner can protect the privacy of his/her own data. In addition, to preserve query privacy, authorized users require encrypting their queries before sending them to the cloud for evaluation. Furthermore, during query processing, the cloud can also derive useful and sensitive information about the actual data items by observing the data access patterns even if the data and query are encrypted [4], [5]. Therefore, following from the above discussions, secure query processing needs to guarantee (1) confidentiality of the encrypted data (2) confidentiality of a user’s query record and (3) hiding data access patterns.

Using encryption as a way to achieve data confidentiality may cause another issue during the query processing step in the cloud. In general, it is very difficult to process encrypted data without ever having to decrypt it. The question here is how the cloud can execute the queries over encrypted data while the data stored at the cloud are encrypted at all times. In the literature, various techniques related to query processing over encrypted data have been proposed, including range queries [6]–[8] and other aggregate queries [9], [10]. However, these techniques are either not applicable or inefficient to solve advanced queries such as the $k$-nearest neighbor ($k$NN) query.

In this paper, we address the problem of secure processing of $k$-nearest neighbor query over encrypted data ($S$kNN) in the cloud. Given a user’s input query $Q$, the objective of the $S$kNN problem is to securely identify the $k$-nearest data tuples to $Q$ using the encrypted database of $T$ in the cloud, without allowing the cloud to learn anything regarding the actual contents of the database $T$ and the query record $Q$. More specifically, when encrypted data are outsourced to the cloud, we observe that an effective $S$kNN protocol needs to satisfy the following properties:

- Preserve the confidentiality of $T$ and $Q$ at all times
- Hiding data access patterns from the cloud
- Accurately compute the $k$-nearest neighbors of query $Q$
- Incur low computation overhead on the end-user

In the past few years, researchers have proposed various methods [1], [11]–[13] to address the $S$kNN problem. However, we emphasize that the existing $S$kNN methods violate at least one of the above mentioned desirable properties of a $S$kNN protocol. On one hand, the methods in [1], [11] are insecure because they are vulnerable to chosen and known plaintext attacks. On the other hand, recent method in [13] returns non-accurate $k$NN result to the end-user. More precisely, in [13], the cloud retrieves the relevant encrypted partition instead of finding the encrypted exact $k$-nearest neighbors. Furthermore, in [1], [12], [13], the end-user involves in heavy computations during the query processing step. By doing so, these methods utilize cloud as just a storage medium, i.e., no significant work
is done on the cloud side. Additionally, the existing \( \mathcal{S}kNN \) methods do not protect data access patterns from the cloud. More details about the existing \( \mathcal{S}kNN \) methods are provided in Section II.

Along this direction, with the goal of providing better security, this paper proposes a novel \( \mathcal{S}kNN \) protocol that satisfies the above properties altogether. The protocols developed in this paper are secure under the semi-honest model [14]. However, they can easily be extended to secure protocols under other adversary models, such as malicious and covert, using threshold based cryptosystem and zero-knowledge proofs.

A. Problem Definition

Suppose the data owner Alice owns a database \( T \) of \( n \) records, denoted by \( t_1, \ldots, t_n \), and \( m \) attributes. Let \( t_{i,j} \) denote the \( j^{\text{th}} \) attribute value of record \( t_i \). In our problem setting, we assume that Alice initially encrypts her database attribute-wise, that is, she computes \( E_{pk}(t_{i,j}) \), for \( 1 \leq i \leq n \) and \( 1 \leq j \leq m \), where \( E_{pk} \) denotes the encryption function of a public-key cryptosystem that is semantically secure [15]. Let the encrypted database be denoted by \( E_{pk}(T) \). We assume that Alice outsources \( E_{pk}(T) \) as well as the future querying processing services to the cloud.

Consider an authorized user Bob who wants to ask the cloud for \( k \)-neighbor records that are closest to his input query \( Q = \langle q_1, \ldots, q_m \rangle \) based on \( E_{pk}(T) \). During this process, Bob’s query \( Q \) and contents of database \( T \) should not be revealed to the cloud. In addition, the access patterns to the data should be protected from the cloud. We refer to such a process as Secure \( kNN \) (\( \mathcal{S}kNN \)) query over encrypted data in the cloud. Without loss of generality, let \( \langle t'_{1}, \ldots, t'_{k} \rangle \) denote the \( k \)-nearest records to \( Q \). Then, we formally define the \( \mathcal{S}kNN \) protocol as follows:

\[
\mathcal{S}kNN(E_{pk}(T), Q) \rightarrow \langle t'_{1}, \ldots, t'_{k} \rangle
\]

We emphasize that, at the end of the \( \mathcal{S}kNN \) protocol, the output \( \langle t'_{1}, \ldots, t'_{k} \rangle \) should be revealed only to Bob. We now present a real-life application of the \( \mathcal{S}kNN \) protocol.

Example 1: Consider a physician who wants to know the risk factor of heart disease in a specific patient. Let \( T \) denote the sample heart disease dataset with attributes \( \text{record-id}, \text{age}, \text{sex}, \text{cp}, \text{trestbps}, \text{chol}, \text{fbs}, \text{slope}, \text{ca}, \text{thal}, \) and \( \text{num} \) as shown in Table I. The heart disease dataset given in Table I is obtained from the UCI machine learning repository [16].

Initially, the data owner (hospital) encrypts \( T \) attribute-wise, outsources the encrypted database \( E_{pk}(T) \) to the cloud for easy management. In addition, the data owner delegates the future query processing services to the cloud. Now, we consider a doctor working at the hospital, say Bob, who would like to know the risk factor of heart disease in a specific patient based on \( T \). Let the patient medical information be \( Q = \langle 58, 1, 4, 133, 196, 1, 2, 1, 6 \rangle \). In the \( \mathcal{S}kNN \) protocol, Bob first need to encrypt \( Q \) (to preserve the privacy of his query) and send it to the cloud. Then the cloud searches on the encrypted database \( E_{pk}(T) \) to figure out the \( k \)-nearest neighbors to the user’s request. For simplicity, let us assume \( k = 2 \). Under this case, the 2 nearest neighbors to \( Q \) are \( t_4 \) and \( t_5 \) (by using Euclidean distance as the similarity metric). After this, the cloud sends \( t_4 \) and \( t_5 \) (in encrypted form) to Bob. Here, the cloud should identify the nearest neighbors of \( Q \) in an oblivious manner without knowing any sensitive information, i.e., all the computations have to be carried over encrypted records. Finally, Bob receives \( t_4 \) and \( t_5 \) that will help him to make medical decisions.

B. Our Contribution

In this paper, we propose a novel \( \mathcal{S}kNN \) protocol to facilitate the \( k \)-nearest neighbor search over encrypted data in the cloud that preserves both the data privacy and query privacy. In our protocol, once the encrypted data are outsourced to the cloud, Alice does not participate in any computations. Therefore, no information is revealed to Alice. In particular, the proposed protocol meets the following requirements:

- **Data confidentiality** - Contents of \( T \) or any intermediate results should not be revealed to the cloud.
- **Query privacy** - Bob’s input query \( Q \) should not be revealed to the cloud.
- **Correctness** - The output \( \langle t'_{1}, \ldots, t'_{k} \rangle \) should be revealed only to Bob. In addition, no information other than \( t'_{1}, \ldots, t'_{k} \) should be revealed to Bob.
- **Low computation overhead on Bob** - After sending his encrypted query record to the cloud, our protocols incur low computation overhead on Bob compared with the existing works [1]. [11]–[13].
- **Hidden data access patterns** - Access patterns to the data, such as the records corresponding to the \( k \)-nearest neighbors of \( Q \), should not be revealed to Alice and the cloud (to prevent any inference attacks).

We emphasize that the intermediate results seen by the cloud in our protocol are either newly generated randomized encryptions or random numbers. Thus, which data records correspond to the \( k \)-nearest neighbors of \( Q \) are not known to the cloud. Also, after sending his encrypted query record to the cloud, Bob does not involve in any computations (low cost on Bob).

Hence, data access patterns are further protected from Bob.

The rest of the paper is organized as follows. We discuss the existing related work and some background concepts in Section II. A set of security primitives that are utilized in the proposed protocols and their possible implementations are provided in Section III. The proposed protocols are explained in detail in Section IV. Section V discusses the performance of the proposed protocols based on various experiments. We conclude the paper along with future work in Section VI.

### Table I

<table>
<thead>
<tr>
<th>record-id</th>
<th>age</th>
<th>sex</th>
<th>cp</th>
<th>trestbps</th>
<th>chol</th>
<th>fbs</th>
<th>slope</th>
<th>ca</th>
<th>thal</th>
<th>num</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t_1 )</td>
<td>63</td>
<td>1</td>
<td>1</td>
<td>145</td>
<td>233</td>
<td>1</td>
<td>3</td>
<td>0</td>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>( t_2 )</td>
<td>56</td>
<td>1</td>
<td>3</td>
<td>130</td>
<td>256</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>( t_3 )</td>
<td>57</td>
<td>0</td>
<td>3</td>
<td>140</td>
<td>241</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td>( t_4 )</td>
<td>59</td>
<td>1</td>
<td>4</td>
<td>144</td>
<td>200</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>( t_5 )</td>
<td>55</td>
<td>0</td>
<td>4</td>
<td>128</td>
<td>205</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>7</td>
<td>3</td>
</tr>
</tbody>
</table>
II. RELATED WORK AND BACKGROUND

In this section, we present an overview of the existing secure k-nearest neighbor techniques. Then, we discuss the security definition adopted in this paper along with the homomorphic properties of the Paillier cryptosystem as a background.

A. Existing Sk\text{NN} Techniques

Retrieving the k-nearest neighbors to a given query \( Q \) is one of the most fundamental problems in many application domains such as similarity search, pattern recognition, and data mining. In the literature, many techniques have been proposed to address the Sk\text{NN} problem, which can be classified into two categories based on whether the data are encrypted or not: centralized and distributed.

1) Centralized Methods: In the centralized methods, we assume that the data owner outsources his/her database and DBMS functionalities (e.g., kNN query) to an untrusted external service provider which manages the data on behalf of the data owner where only trusted users are allowed to query the hosted data. By outsourcing data to an untrusted server, many security issues arise, such as data privacy (protecting the confidentiality of the data from the server and query issuer). To achieve data privacy, data owner is required to use data anonymization models (e.g., \( k \)-anonymity) or cryptographic (e.g., encryption and data perturbation) techniques over his/her data before outsourcing them to the server.

Encryption is a traditional technique used to protect the confidentiality of sensitive data such as medical records. Due to data encryption, the process of query evaluation over encrypted data becomes challenging. Along this direction, various techniques have been proposed for processing range [6]–[8] and aggregation queries [9], [10] over encrypted data. However, in this paper, we restrict our discussion to secure evaluation of kNN query.

In the past few years, researchers have proposed different methods [1], [11]–[13] to address the Sk\text{NN} problem. Wong et al. [11] proposed a new encryption scheme called asymmetric scalar-product-preserving encryption (ASPE) that preserves scalar product between the query vector \( Q \) and any tuple vector \( t_i \) from database \( T \) for distance comparison which is sufficient to find kNN. In [11], data and query are encrypted using slightly different encryption schemes before outsourcing to the server and all the query users know the decryption key. As an improvement, Zhu et al. [12] proposed a novel Sk\text{NN} method in which the key of the data owner is not disclosed to the user. However, their architecture requires the participation of data owner during query encryption. As an alternative, Hu et al. [1] proposed a method based on provably secure privacy homomorphism encryption scheme from [17] that supports modular addition, subtraction and multiplication over encrypted data. They addressed the Sk\text{NN} problem under the following setting: the client has the ciphertexts of all data points in database \( T \) and the encryption function of \( T \) whereas the server has the decryption function of \( T \) and some auxiliary information regarding each data point. However, both methods in [1], [11] are not secure because they are vulnerable to chosen-plaintext attacks. Also, all the above methods leak data access patterns to the server.

Recently, Yao et al. [13] proposed a new Sk\text{NN} method based on partition-based secure Voronoi diagram (SVD). Instead of asking the cloud to retrieve the exact kNN, they required, from the cloud, to retrieve a relevant encrypted partition \( E_{pk}(G) \) for \( E_{pk}(T) \) such that \( G \) is guaranteed to contain the k-nearest neighbors of \( Q \). However, in our work, we are able to solve the Sk\text{NN} problem accurately by letting the cloud to retrieve the exact k-nearest neighbors of \( Q \) (in encrypted form). In addition, most of the computations during the query processing step in [1], [12], [13] are performed locally by the end-user which conflicts the very purpose of outsourcing the DBMS functionalities to the cloud. Furthermore, the protocol in [13] leaks data access patterns, such as the partition ID corresponding to a user query, to the cloud.

2) Data Distribution Methods: In the data distribution methods, data are assumed to be partitioned either vertically or horizontally and distributed among a set of independent, non-colluding parties. In the literature, the data distribution methods rely on secure multiparty computation (SMC) techniques that enable multiple parties to securely evaluate a function using their respective private inputs without disclosing the input of one party to the others. Many efforts have been made to address the problem of kNN query in a distributed environment. Shaneck et al. [18] proposed privacy-preserving algorithm to perform k-nearest neighbor search. The protocol in [18] is based on secure multiparty computation for privately computing kNN points in a horizontally partitioned dataset. Qi et al. [19] proposed a single-step kNN search protocol that is provably secure with linear computation and communication complexities. Vaidya et al. [20] studied privacy-preserving top-k queries in which the data are vertically partitioned. Ghinita et al. [21] proposed a private information retrieval (PIR) based framework for answering kNN queries in location-based services. We emphasize that, in [21], the data residing at the server are in plaintext format. However, if the data are encrypted to ensure data confidentiality, it is not clear how a user can obliviously retrieve the output records because he/she does not know the indexes that match his/her input query. Nevertheless, even if a user can retrieve the records using PIR, the user still needs to perform local computations to identify the k-nearest neighbors. However, in our framework, the users computation is completely outsourced to a cloud.

In summary, we emphasize that the above data distribution methods are not applicable to perform kNN queries over encrypted data for two reasons: (1). In our work, we deal with encrypted form of database and query which is not the case in the above methods (2). The database in our case is encrypted and stored on the cloud whereas in the above methods it is partitioned (in plaintext format) among different parties.

B. Security Definition

In this paper, privacy/security is closely related to the amount of information disclosed during the execution of a protocol. There are many ways to define information disclo-
sure. To maximize privacy or minimize information disclosure, we adopt the security definitions in the literature of secure multiparty computation (SMC) first introduced by Yao’s Millionaires’ problem for which a provably secure solution was developed [14]. In this paper, we assume that parties are semi-honest (or honest-but-curious); that is, a semi-honest party follows the rules of the protocol using its correct input, but is free to later use what it sees during execution of the protocol to compromise security. In general, secure protocols under the semi-honest model are more efficient than those under the malicious adversary model, and almost all practical SMC protocols proposed in the literature are secure under the semi-honest model. Due to space limitations, we refer the reader to [14] for detailed security definitions and models.

### C. Paillier Cryptosystem

The Paillier cryptosystem is an additive homomorphic and probabilistic asymmetric encryption scheme [15]. Let $E_{pk}$ be the encryption function with public key $pk$ given by $(N, g)$, where $N$ is a product of two large primes and $g$ is in $\mathbb{Z}_N^\times$. Also, let $D_{sk}$ be the decryption function with secret key $sk$.

Given $a, b \in \mathbb{Z}_N$, the Paillier encryption scheme exhibits the following properties:

- **Homomorphic Addition**
  
  $$E_{pk}(a + b) \leftarrow E_{pk}(a) \cdot E_{pk}(b) \mod N^2;$$

- **Homomorphic Multiplication**
  
  $$E_{pk}(a \cdot b) \leftarrow E_{pk}(a)^b \mod N^2;$$

- **Semantic Security** - The encryption scheme is semantically secure [22], i.e., given a set of ciphertexts, an adversary cannot deduce any information about the plaintext.

In this paper, we assume that a data owner encrypted his or her data using Paillier cryptosystem before outsourcing them to a cloud. Some common notations that are used extensively in this paper are shown in Table II.

### III. BASIC SECURITY PRIMITIVES

In this section, we present a set of generic protocols that will be used as sub-routines while constructing our proposed Sk:NN protocol in Section IV-B. All of the below protocols are considered under two-party semi-honest setting. In particular, we assume the existence of two semi-honest parties $P_1$ and $P_2$ such that the Paillier’s secret key $sk$ is known only to $P_2$ whereas $pk$ is treated as public.

- **Secure Multiplication (SM) Protocol:** This protocol considers $P_1$ with input $(E_{pk}(a), E_{pk}(b))$ and outputs $E_{pk}(a \cdot b)$ to $P_1$, where $a$ and $b$ are not known to $P_1$ and $P_2$. During this process, no information regarding $a$ and $b$ is revealed to $P_1$ and $P_2$. The output $E_{pk}(a \cdot b)$ is known only to $P_1$.

- **Secure Squared Euclidean Distance (SSED) Protocol:** $P_1$ with input $(E_{pk}(X), E_{pk}(Y))$ and $P_2$ securely compute the encryption of squared Euclidean distance between vectors $X$ and $Y$. Here $X$ and $Y$ are $m$ dimensional vectors where $E_{pk}(X) = \{E_{pk}(x_1), \ldots, E_{pk}(x_m)\}$ and $E_{pk}(Y) = \{E_{pk}(y_1), \ldots, E_{pk}(y_m)\}$. At the end, the output $E_{pk}((X - Y)^2)$ is known only to $P_1$.

- **Secure Bit-Decomposition (SBD) Protocol:** $P_1$ with input $E_{pk}(z)$ and $P_2$ securely compute the encryptions of the individual bits of $z$, where $0 \leq z < 2^l$. The output $[z] = \{E_{pk}(z_1), \ldots, E_{pk}(z_l)\}$ is known only to $P_1$. Here $z_1$ and $z_2$ denote the most and least significant bits of integer $z$ respectively.

- **Secure Minimum (SMIN) Protocol:** $P_1$ with input $([a], [v])$ and $P_2$ with $sk$ securely compute the encryptions of the individual bits of minimum number between $u$ and $v$. That is, the output is $\min(u, v)$ which will be known only to $P_1$. During this protocol, no information regarding $u$ and $v$ is revealed to $P_1$ and $P_2$.

- **Secure Minimum out of $n$ Numbers (SMIN$_n$) Protocol:** $P_1$ has $n$ encrypted vectors $([d_1], \ldots, [d_n])$ and $P_2$ has $sk$. Here $[d_i] = \{E_{pk}(d_{i1}), \ldots, E_{pk}(d_{il})\}$ such that $d_{i1}$ and $d_{il}$ are the most and least significant bits of integer $d_i$, respectively, for $1 \leq i \leq n$. $P_1$ and $P_2$ jointly compute the output $\min(d_{11}, \ldots, d_{n1})$. At the end, $\min(d_{11}, \ldots, d_{n1})$ is known only to $P_1$. During SMIN$_n$, no information about $d_{i1}$’s is revealed to $P_1$ and $P_2$.

- **Secure Bit-OR (SBOR) Protocol:** $P_1$ with input $E_{pk}(o_1), E_{pk}(o_2)$ and $P_2$ securely compute $E_{pk}(o_1 \lor o_2)$, where $o_1$ and $o_2$ are two bits. The output $E_{pk}(o_1 \lor o_2)$ is known only to $P_1$.

We now discuss each of these protocols in detail. Also, we either propose new solution or refer to the most efficient known implementation to each one of them.

### Secure Multiplication (SM). Consider a party $P_1$ with private input $(E_{pk}(a), E_{pk}(b))$ and a party $P_2$ with the secret key $sk$. The goal of the secure multiplication (SM) protocol is to return the encryption of $a \cdot b$, i.e., $E_{pk}(a \cdot b)$ as output to $P_1$.

During this protocol, no information regarding $a$ and $b$ is revealed to $P_1$ and $P_2$. The basic idea of SM is based on the following property which holds for any given $a, b \in \mathbb{Z}_N$:

$$a \cdot b = (a + r_a) \cdot (b + r_b) - a \cdot r_b - b \cdot r_a - r_a \cdot r_b \quad (1)$$

where all the arithmetic operations are performed under $\mathbb{Z}_N$. The overall steps in SM are shown in Algorithm 1. Briefly, $P_1$ initially randomizes $a$ and $b$ by computing $a' = E_{pk}(a) \cdot E_{pk}(r_a)$ and $b' = E_{pk}(b) \cdot E_{pk}(r_b)$, and sends them to $P_2$. Here $r_a$ and $r_b$ are random numbers in

<table>
<thead>
<tr>
<th>Table II: Common Notations</th>
</tr>
</thead>
<tbody>
<tr>
<td>$E_{pk}(T)$</td>
</tr>
<tr>
<td>$(n, m)$</td>
</tr>
<tr>
<td>$(t, Q)$</td>
</tr>
<tr>
<td>$T_i'$</td>
</tr>
<tr>
<td>$l$</td>
</tr>
<tr>
<td>$(z_1, z_2)$</td>
</tr>
<tr>
<td>$[z]$</td>
</tr>
</tbody>
</table>
Algorithm 1 SM($E_{pk}(a), E_{pk}(b)) \rightarrow E_{pk}(a \ast b)$

Require: $P_1$ has $E_{pk}(a)$ and $E_{pk}(b)$; $P_2$ has $sk$

1: $P_1$
   (a) Pick two random numbers $r_a, r_b \in \mathbb{Z}_N$
   (b) $a' \leftarrow E_{pk}(a) \ast E_{pk}(r_a)$
   (c) $b' \leftarrow E_{pk}(b) \ast E_{pk}(r_b)$; send $a', b'$ to $P_2$

2: $P_2$
   (a) Receive $a'$ and $b'$ from $P_1$
   (b) $h_a \leftarrow D_{sk}(a')$; $h_b \leftarrow D_{sk}(b')$
   (c) $h \leftarrow h_a \ast h_b \mod N$
   (d) $h' \leftarrow E_{pk}(h)$; send $h'$ to $P_1$

3: $P_1$
   (a) Receive $h'$ from $P_2$
   (b) $s \leftarrow h' \ast E_{pk}(a)^{N-r_a}$
   (c) $s' \leftarrow s \ast E_{pk}(b)^{N-r_b}$
   (d) $E_{pk}(a \ast b) \leftarrow s' \ast E_{pk}(r_a \ast r_b)^{N-1}$

$\mathbb{Z}_N$ known only to $P_1$. Upon receiving, $P_2$ decrypts and multiplies them to get $h = (a + r_a) \ast (b + r_b) \mod N$. Then, $P_2$ encrypts $h$ and sends it to $P_1$. After this, $P_3$ removes extra random factors from $h' = E_{pk}(h(a + r_a) \ast (b + r_b))$ based on Equation 1 to get $E_{pk}(a \ast b)$. Note that, for any given $x \in \mathbb{Z}_N$, “$N - x$” is equivalent to “$-x$” under $\mathbb{Z}_N$. Hereafter, we use the notation $r \in R \mathbb{Z}_N$ to denote $r$ as a random number in $\mathbb{Z}_N$.

Secure Squared Euclidean Distance (SSED). Suppose $P_1$ holds two encrypted vectors ($E_{pk}(X), E_{pk}(Y)$) and $P_2$ holds $sk$. Here $X$ and $Y$ are two $m$-dimensional vectors such that $E_{pk}(X) = \langle E_{pk}(x_1), \ldots, E_{pk}(x_m) \rangle$ and $E_{pk}(Y) = \langle E_{pk}(y_1), \ldots, E_{pk}(y_m) \rangle$. The goal of SSED is to securely compute $E_{pk}(|X - Y|^2)$, where $|X - Y|$ denotes the Euclidean distance between $X$ and $Y$. During this protocol, no information regarding $X$ and $Y$ is revealed to $P_1$ and $P_2$. The basic idea of SSED follows from the following equation:

$$|X - Y|^2 = \sum_{i=1}^{m} (x_i - y_i)^2 \quad (2)$$

The main steps involved in SSED are shown in Algorithm 2. Briefly, for $1 \leq i \leq m$, $P_1$ initially computes $E_{pk}(x_i - y_i)$ by using the homomorphic properties. Then $P_1$ and $P_2$ jointly compute $E_{pk}((x_i - y_i)^2)$ using the SM protocol, for $1 \leq i \leq m$. Note that the outputs of SM are known only to $P_1$. By applying homomorphic properties on $E_{pk}((x_i - y_i)^2)$, $P_1$ computes $E_{pk}(|X - Y|^2)$ locally based on Equation 2.

Secure Bit-Decomposition (SBD). Suppose $P_1$ has $E_{pk}(z)$ and $P_2$ has $sk$, where $z$ is not known to both parties and $0 \leq z < 2^l$. The goal of SBD is to compute the encryptions of the individual bits of binary representation of $z$ [23]. The output is $[z] = \langle E_{pk}(z_1), \ldots, E_{pk}(z_l) \rangle$, where $z_1$ and $z_l$ denote the most and least significant bits of $z$ respectively. At the end, the output $[z]$ is known only to $P_1$. Since the goal of this paper is to investigate the existing SBD protocols, we simply use the most efficient SBD protocol that was recently proposed in [23].

Algorithm 2 SSED($E_{pk}(X), E_{pk}(Y)) \rightarrow E_{pk}(|X - Y|^2)$

Require: $P_1$ has $E_{pk}(X)$ and $E_{pk}(Y)$; $P_2$ has $sk$

1: $P_1$, for $1 \leq i \leq m$
   (a) $E_{pk}(x_i - y_i) \leftarrow E_{pk}(x_i) \ast E_{pk}(y_i)^{N-1}$

2: $P_1$ and $P_2$, for $1 \leq i \leq m$
   (a) Compute $E_{pk}((x_i - y_i)^2)$ using the SM protocol

3: $P_1$ computes $E_{pk}(|X - Y|^2) \leftarrow \prod_{i=1}^{m} E_{pk}(x_i - y_i)^2$

Secure Minimum (SMIN). In this protocol, $P_1$ with input $([u], [v])$ and $P_2$ with $sk$ securely compute the encryptions of the individual bits of min$(u, v)$, i.e., the output is $[\min(u, v)]$. Here $[u] = \langle E_{pk}(u_1), \ldots, E_{pk}(u_l) \rangle$ and $[v] = \langle E_{pk}(v_1), \ldots, E_{pk}(v_l) \rangle$, where $u_1$ (resp., $v_1$) and $u_l$ (resp., $v_l$) are the most and least significant bits of $u$ (resp., $v$). At the end, the output $[\min(u, v)]$ is known only to $P_1$.

We assume that $0 \leq u, v < 2^l$ and propose a novel SMIN protocol. The basic idea of the proposed SMIN protocol is for $P_1$ to randomly choose the functionality $F$ (by flipping a coin), where $F$ is either $u > v$ or $v > u$, and to obliviously execute $F$ with $P_2$. Since $F$ is randomly chosen and known only to $P_1$, the output of the functionality $F$ is oblivious to $P_2$. Based on the output and chosen $F$, $P_1$ computes $[\min(u, v)]$ locally using homomorphic properties.

The overall steps involved in the SMIN protocol are shown in Algorithm 3. To start with, $P_1$ initially chooses the functionality $F$ as either $u > v$ or $v > u$ randomly. Then, using SM, $P_1$ computes $E_{pk}(u_i \ast v_i)$ with the help of $P_2$. Now, depending on $F$, $P_1$ proceeds as follows, for $1 \leq i \leq l$:

- If $F : u > v$, compute:
  $$W_i = E_{pk}(u_i) \ast E_{pk}(u_i \ast v_i)^{N-1} = E_{pk}(u_i \ast (1 - v_i))$$
  $$\Gamma_i = E_{pk}(u_i - u_i) \ast E_{pk}(\hat{r}_i) = E_{pk}(u_i - u_i + \hat{r}_i)$$

- If $F : v > u$, compute:
  $$W_i = E_{pk}(v_i) \ast E_{pk}(u_i \ast v_i)^{N-1} = E_{pk}(v_i \ast (1 - u_i))$$
  $$\Gamma_i = E_{pk}(u_i - u_i) \ast E_{pk}(\hat{r}_i) = E_{pk}(u_i - u_i + \hat{r}_i)$$

where $\hat{r}_i$ is a random number in $\mathbb{Z}_N$.

Observe that if $F : u > v$, then $W_i = E_{pk}(1)$ only if $u_i > v_i$, and $W_i = E_{pk}(0)$ otherwise. Similarly, when $F : v > u$, we have $W_i = E_{pk}(1)$ only if $v_i > u_i$, and $W_i = E_{pk}(0)$ otherwise. Also, depending of $F$, $\Gamma_i$, stores the encryption of randomized difference between $u_i$ and $v_i$ which will be used in later computations.

- Compute the encrypted bit-wise XOR between the bits $u_i$ and $v_i$ as $G_i = E_{pk}(u_i \oplus v_i)$ using the below formulation:
  $$G_i = E_{pk}(u_i) \ast E_{pk}(v_i) \ast E_{pk}(u_i \ast v_i)^{N-2}$$

In general, for any two given bits $o_1$ and $o_2$, we have $o_1 \oplus o_2 = o_1 + o_2 - 2(o_1 \ast o_2)$

- Compute an encrypted vector $H$ by preserving the first occurrence of $E_{pk}(1)$ (if there exists one) in $G$ by initializing $H_0 = E_{pk}(0)$. The rest of the entries of $H$ are computed as $H_i = H_{i-1} \ast G_i$. We emphasize that at most one of the entry in $H$ is $E_{pk}(1)$ and the remaining
Algorithm 3 $\text{SMIN}([u], [v]) \rightarrow [\min(u, v)]$

Require: $P_1$ has $[u]$ and $[v]$, where $0 \leq u, v < 2^k$; $P_2$ has $sk$

1: $P_1$:
   (a) Randomly choose the functionality $F$
   (b) for $i = 1$ to $l$ do:
      • $E_{pk}(u_i \ast v_i) \leftarrow \text{SM}(E_{pk}(u_i), E_{pk}(v_i))$
      • if $F : u > v$ then:
         - $W_i \leftarrow E_{pk}(u_i) \ast E_{pk}(u_i \ast v_i)^{N-1}$
         - $\Gamma_{i} \leftarrow E_{pk}(v_i - u_i) \ast E_{pk}(r_i); \ r_i \in R \ Z_N$
      else:
         - $W_i \leftarrow E_{pk}(v_i) \ast E_{pk}(u_i \ast v_i)^{N-1}$
         - $\Gamma_i \leftarrow E_{pk}(u_i - v_i) \ast E_{pk}(r_i); \ r_i \in R \ Z_N$
         • $G_i \leftarrow E_{pk}(u_i \oplus v_i)$
         • $H_i \leftarrow H_i^{\Gamma - 1} \ast G_i; \ r_i \in R \ Z_N \text{ and } H_0 = E_{pk}(0)$
         • $\Phi_i \leftarrow E_{pk}(-1) \ast H_i$
         • $L_i \leftarrow W_i \ast \Phi_i^{\prime}; \ r_i \ast R \ Z_N$
   (c) $\Gamma' \leftarrow \pi_1(\Gamma)$
   (d) $L' \leftarrow \pi_2(L)$; send $\Gamma'$ and $L'$ to $P_2$

2: $P_2$:
   (a) Receive $\Gamma'$ and $L'$ from $P_1$
   (b) $M_i \leftarrow D_{sk}(L_i')$, for $1 \leq i \leq l$
   (c) if $\exists j$ such that $M_j = 1$ then $\alpha \leftarrow 1$
      else $\alpha \leftarrow 0$
   (d) $M_i' \leftarrow \Gamma_i^{\alpha}$, for $1 \leq i \leq l$
   (e) Send $M'$ and $E_{pk}(\alpha)$ to $P_1$

3: $P_1$:
   (a) Receive $M'$ and $E_{pk}(\alpha)$ from $P_2$
   (b) $\bar{M} \leftarrow \pi_1^{-1}(M')$
   (c) for $i = 1$ to $l$ do:
      • $\lambda_i \leftarrow \bar{M}_i \ast E_{pk}(\alpha)^{N-r_i}$
      • if $F : u > v$ then $E_{pk}(\min(u, v), i) \leftarrow E_{pk}(u_i) \ast \lambda_i$
      else $E_{pk}(\min(u, v), i) \leftarrow E_{pk}(v_i) \ast \lambda_i$

entries are encryptions of either 0 or a random number. Also, if there exists an index $j$ such that $H_j = E_{pk}(1)$, then index $j$ is the first position (from the most significant bit) at which the corresponding bits of $u$ and $v$ differ.

Then, $P_1$ computes $\Phi_i = E_{pk}(-1) \ast H_i$. Note that “−1” is equivalent to “$N − 1$” under $Z_N$. From the above discussions, it is clear that $\Phi_i = E_{pk}(0)$ at most one since $H_i$ is equal to $E_{pk}(1)$ at most once. Also, if $\Phi_j = E_{pk}(0)$, then index $j$ is the position at which the bits of $u$ and $v$ differ first.

Compute an encrypted vector $L$ by combining $W$ and $\Phi$. Note that $W_i$ stores the result of $u_i > v_i$ or $v_i > u_i$ which depends on $F$ known only to $P_1$. Precisely, $P_1$ computes $L_i = W_i \ast \Phi_i^{\prime}$, where $r_i'$ is a random number in $Z_N$. The observation here is if $\exists$ an index $j$ such that $\Phi_j = E_{pk}(0)$, denoting the first flip in the bits of $u$ and $v$, then $W_j$ stores the corresponding desired information, i.e., whether $u_j > v_j$ or $v_j > u_j$ in encrypted form.

After this, $P_1$ permutes the encrypted vectors $\Gamma$ and $L$ using two random permutation functions $\pi_1$ and $\pi_2$. Specifically,

<table>
<thead>
<tr>
<th>$[u]$</th>
<th>$[v]$</th>
<th>$W_i$</th>
<th>$\Gamma_i$</th>
<th>$H_i$</th>
<th>$\Phi_i$</th>
<th>$L_i$</th>
<th>$L_i'$</th>
<th>$M_i$</th>
<th>$\lambda_i$</th>
<th>$\min_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>r</td>
<td>0</td>
<td>0</td>
<td>−1</td>
<td>r</td>
<td>1</td>
<td>r</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1+1</td>
<td>r</td>
<td>r</td>
<td>0</td>
<td>r</td>
<td>1</td>
<td>r</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>r</td>
<td>1+1</td>
<td>r</td>
<td>0</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1+1</td>
<td>r</td>
<td>r</td>
<td>0</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>r</td>
<td>1+1</td>
<td>r</td>
<td>0</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1+1</td>
<td>r</td>
<td>0</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1+1</td>
<td>r</td>
<td>0</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

All column values are in encrypted form ($E_{pk}(0)$) except $M_i$ column. Also, $r$ is a random in $Z_N$ which is different for each row and column. $P_1$ computes $\Gamma' = \pi_1(\Gamma)$ and $L' = \pi_2(L)$, and sends them to $P_2$. Upon receiving, $P_2$ decrypts $L'$ component-wise to get $M_i = D_{sk}(L_i')$, for $1 \leq i \leq l$, and checks for index $j$ (decide the output of $F$). That is, if $M_j = 1$, then the output of $F$ is 1, and 0 otherwise. Let the output be $\alpha$. Note that since $F$ is not known to $P_2$, the output $\alpha$ is oblivious to $P_2$. In addition, $P_2$ computes a new encrypted vector $M'$ where $M'_i = \Gamma_i^{\alpha}$, for $1 \leq i \leq l$, sends $M'$ and $E_{pk}(\alpha)$ to $P_1$. After receiving $M'$ and $E_{pk}(\alpha)$, $P_1$ computes the inverse permutation of $M'$ as $\bar{M} = \pi_1^{-1}(M')$. Then, $P_1$ performs the following homomorphic operations to compute the encryption of $i^{th}$ bit of $\min(u, v)$, i.e., $E_{pk}(\min(u, v), i)$, for $1 \leq i \leq l$:

- Remove the randomness from $\bar{M}_i$ by computing $\lambda_i = \bar{M}_i \ast E_{pk}(\alpha)^{N-r_i}$
- If $F : u > v$, compute the $i^{th}$ encrypted bit of $\min(u, v)$ as $E_{pk}(\min(u, v), i) = E_{pk}(u_i) \ast \lambda_i = E_{pk}(u_i + \alpha \ast (v_i - u_i))$. Otherwise, compute $E_{pk}(\min(u, v), i) = E_{pk}(v_i) \ast \lambda_i = E_{pk}(v_i + \alpha \ast (u_i - v_i))$.

In the SMIN protocol, one main observation (upon which we can also justify the correctness of the final output) is that if $F : u > v$, then $\min(u, v, i) = (1 - \alpha) \ast u_i + \alpha \ast v_i$ always holds, for $1 \leq i \leq l$. Similarly, if $F : v > u$, then $\min(u, v, i) = \alpha \ast u_i + (1 - \alpha) \ast v_i$ always holds.

**Example 2:** Consider that $u = 55$, $v = 58$, and $l = 6$. We assume that $P_1$’s random permutation functions are given as below. Suppose $P_1$ holds

$$i = 1 2 3 4 5 6$$

$$\pi_1(i) = 6 5 4 3 2 1$$

$$\pi_2(i) = 2 1 5 6 3 4$$

$[55] = (E_{pk}(1), E_{pk}(1), E_{pk}(0), E_{pk}(1), E_{pk}(1), E_{pk}(1))$ and $[58] = (E_{pk}(1), E_{pk}(1), E_{pk}(1), E_{pk}(0), E_{pk}(1), E_{pk}(0))$. Without loss of generality, suppose $P_1$ chooses the functionality $F : v > u$. Then, various intermediate results based on the SMIN protocol are as shown in Table III. Following from Table III, we observe that:

- At most one of the entry in $H$ is $E_{pk}(1) (= H_3)$ and the remaining entries are encryptions of either 0 or a random number in $Z_N$. Index $j = 3$ is the first position at which the corresponding bits of $u$ and $v$ differ.
- $\Phi_3 = E_{pk}(0)$ since $H_3$ is equal to $E_{pk}(1)$. Also, since $M_3 = 1$, $P_2$ sets $\alpha$ to 1.

At the end, only $P_1$ knows $[\min(u, v)] = [u] = [55]$. □
Require: $P_1$ has $([d_1],\ldots,[d_n])$; $P_2$ has $sk$

1: $P_1$:  
(a) $[d'_i] \leftarrow [d_i]$, for $1 \leq i \leq n$, and $num \leftarrow n$

2: $P_1$ and $P_2$, for $i = 1$ to $\lceil \log_2 n \rceil$:
(a) for $1 \leq j \leq \lfloor \frac{num}{2} \rfloor$:
   * if $i = 1$ then:
     - $[d'_{2j-1}] \leftarrow \text{SMIN}([d_{2j-1}], [d_{2j}])$
     - $[d'_{2j}] \leftarrow 0$
   else
     - $[d'_{2i(j-1)+1}] \leftarrow \text{SMIN}([d'_{2i(j-1)+1}], [d_{2ij-1}])$
     - $[d'_{2ij-1}] \leftarrow 0$

3: $P_1$ sets $[d_{min}]$ to $[d'_1]$

**Secure Minimum out of $n$ Numbers (SMIN$_n$).** Consider $P_1$ with private input $(d_1,\ldots,d_n)$ and $P_2$ with $sk$, where $0 \leq d_i < 2^l$ and $[d_i] = (E_{pk}(d_{i,1}),\ldots,E_{pk}(d_{i,l}))$, for $1 \leq i \leq n$. The goal of the SMIN$_n$ protocol is to compute $[\min(d_1,\ldots,d_n)] = [d_{min}]$ without revealing any information about $d_i$'s to $P_1$ and $P_2$. Here we construct a new SMIN$_n$ protocol by utilizing SMIN as the building block. The proposed SMIN$_n$ protocol is an iterative approach and it computes the desired output in an hierarchical fashion. In each iteration, minimum between a pair of values is computed and are fed as input to the next iteration. Therefore, generating a binary execution tree in a bottom-up fashion. At the end, only $P_1$ knows the final result $[d_{min}]$.

The overall steps involved in the proposed SMIN$_n$ protocol are highlighted in Algorithm 4. Initially, $P_1$ assigns $[d_i]$ to a temporary vector $[d'_i]$, for $1 \leq i \leq n$. Also, he/she creates a global variable $num$ and initialize it to $n$, where $num$ represents the number of (non-zero) vectors involved in each iteration. Since the SMIN$_n$ protocol executes in a binary tree hierarchy (bottom-up fashion), we have $\lceil \log_2 n \rceil$ iterations, and in each iteration, the number of vectors involved varies.

In the first iteration (i.e., $i = 1$), $P_1$ with private input $([d'_{2j-1}], [d_{2j}])$ and $P_2$ with $sk$ involve in the SMIN protocol, for $1 \leq j \leq \lfloor \frac{num}{2} \rfloor$. At the end of the first iteration, only $P_1$ knows $[\min(d'_{2j-1}, d_{2j})]$ and nothing is revealed to $P_2$, for $1 \leq j \leq \lfloor \frac{num}{2} \rfloor$. Also, $P_1$ stores the result $[\min(d'_{2j-1}, d_{2j})]$ in $[d'_{2j-1}]$ to zero and $num$ to $\lfloor \frac{num}{2} \rfloor$.

During the $i^{th}$ iteration, only the non-zero vectors are involved, for $2 \leq i \leq \lceil \log_2 n \rceil$. For example, during second iteration (i.e., $i = 2$), only $[d'_1], [d'_2]$, and so on are involved. Note that in each iteration, the output is revealed only to $P_1$ and $num$ is updated to $\lfloor \frac{num}{2} \rfloor$. At the end of SMIN$_n$, $P_1$ assigns the final encrypted binary vector of global minimum value, i.e., $[\min(d_1,\ldots,d_n)]$ which is stored in $[d'_1]$ to $[d_{min}]$. For example, assume that $P_1$ holds $(d_1,\ldots,d_6)$ (i.e., $n = 6$). Then, based on the SMIN$_n$ protocol, the binary execution tree (in a bottom-up fashion) to compute $[\min(d_1,\ldots,d_6)]$ is as shown in Figure 1. Note that, $[d'_1]$ is initially set to $[d_i]$, for $1 \leq i \leq 6$.

**Secure Bit-OR (SBOR).** $P_1$ holds $(E_{pk}(o_1), E_{pk}(o_2))$ and $P_2$ holds $sk$, where $o_1$ and $o_2$ are two bits not known to both parties. The goal of the SBOR protocol is to securely compute $E_{pk}(o_1 \lor o_2)$. At the end of this protocol, only $P_1$ knows $E_{pk}(o_1 \lor o_2)$. During this process, no information related to $o_1$ and $o_2$ is revealed to $P_1$ and $P_2$. Using SM, $P_1$ and $P_2$ compute $E_{pk}(o_1 \lor o_2)$ as follows:

- $P_1$ with input $(E_{pk}(o_1), E_{pk}(o_2))$ and $P_2$ with $sk$ involve in the SM protocol. At the end of this step, the output $E_{pk}(o_1 \lor o_2)$ is known only to $P_1$. Note that, since $o_1$ and $o_2$ are bits, $E_{pk}(o_1 \lor o_2) = E_{pk}(o_1 \land o_2)$.
- $E_{pk}(o_1 \lor o_2) = E_{pk}(o_1 + o_2) + E_{pk}(o_1 \land o_2)^{N-1}$.

We emphasize that, for any given two bits $o_1$ and $o_2$, the property $o_1 \lor o_2 = o_1 + o_2 - o_1 \land o_2$ always holds.

It is worth pointing out that SMIN, SMIN$_n$ and SBOR are completely new and are not based on any existing protocols. On the other hand, SSED is not new, but our implementation is more efficient. Also, SM and SBD are directly adopted from the literature.

**IV. THE PROPOSED PROTOCOLS**

In this section, we first present a basic S$k$NN protocol and demonstrate why such a simple solution is not secure. Then, we discuss our second approach, a fully secure $k$NN protocol. Both protocols are constructed using the security primitives discussed in Section III as building blocks.

As mentioned earlier, we assume that Alice’s database consists of $n$ records, denoted by $T = (t_{1,1},\ldots,t_{n,1})$, and $m$ attributes, where $t_{i,j}$ denotes the $j^{th}$ attribute value of record $t_i$. Initially, Alice encrypts her database attribute-wise, that is, she computes $E_{pk}(t_{i,j})$, for $1 \leq i \leq n$ and $1 \leq j \leq m$. Let the encrypted database be denoted by $E_{pk}(T)$. We assume that Alice outsources $E_{pk}(T)$ as well as the future query processing service to the cloud. Also, we assume that all attribute values and their Euclidean distances lie in $[0,2^l]$.

In our proposed protocols, we assume the existence of two non-colluding semi-honest cloud service providers, denoted by $C_1$ and $C_2$, which together form a federated cloud. We emphasize that such an assumption is not new and has been commonly used in the related problem domains (e.g., [24]).

The intuition behind such an assumption is as follows. Most of the cloud service providers in the market are well-established IT companies, such as Amazon and Google. Therefore, a collusion between them is highly unlikely as it will damage their reputation which in turn affects their revenues.
Under this setting, Alice outsources her encrypted database $E_{pk}(T)$ to $C_1$ and the secret key $sk$ to $C_2$. The goal of the proposed protocols is to retrieve the top $k$ records that are closest to the user query in an efficient and secure manner. Briefly, consider an authorized user Bob who wants to find $k$ records that are closest to his query record $Q = \langle q_1, \ldots, q_m \rangle$ based on $E_{pk}(T)$ in $C_1$. Bob initially sends his query $Q$ (in encrypted form) to $C_1$. After this, $C_1$ and $C_2$ involve in a set of sub-protocols to securely retrieve (in encrypted form) the set of $k$ records corresponding to the $k$-nearest neighbors of the input query $Q$. At the end of our protocols, only Bob will receive the $k$-nearest neighbors to $Q$ as the output.

A. Basic Protocol

In the basic secure $k$-nearest neighbor query protocol, denoted by SkNN$_b$, we relax the desirable properties to produce an efficient protocol (more details are given in the later part of this section).

The main steps involved in the SkNN$_b$ protocol are given in Algorithm 5. Bob initially encrypts his query $Q$ attribute-wise, that is, he computes $E_{pk}(Q) = \langle E_{pk}(q_1), \ldots, E_{pk}(q_m) \rangle$ and sends it to $C_1$. Upon receiving $E_{pk}(Q)$ from Bob, $C_1$ with private input $(E_{pk}(Q), E_{pk}(t_i))$ and $C_2$ with the secret key $sk$ jointly involve in the SSED protocol, where $E_{pk}(t_i) = \langle E_{pk}(t_{i,1}), \ldots, E_{pk}(t_{i,m}) \rangle$, for $1 \leq i \leq n$. The output of this step, denoted by $E_{pk}(d_i)$, is the encryption of squared Euclidean distance between $Q$ and $t_i$, i.e., $d_i = |Q - t_i|^2$. As mentioned earlier, $E_{pk}(d_i)$ is known only to $C_1$, for $1 \leq i \leq n$. We emphasize that computation of exact Euclidean distance between encrypted vectors is hard to achieve as it involves square root. However, in our problem, it is sufficient to compare the squared Euclidean distances as it preserves relative ordering. After this, $C_1$ sends $\{\langle 1, E_{pk}(d_1) \rangle, \ldots, \langle n, E_{pk}(d_n) \rangle\}$ to $C_2$, where entry $(i, E_{pk}(d_i))$ correspond to data record $t_i$, for $1 \leq i \leq n$. Upon receiving $\langle 1, E_{pk}(d_1) \rangle, \ldots, \langle n, E_{pk}(d_n) \rangle$, $C_2$ decrypts the encrypted distance in each entry to get $d_i = D_{sk}(E_{pk}(d_i))$. Then, $C_2$ generates an index list $\delta = \langle i_1, \ldots, i_k \rangle$ such that $\langle d_{i_1}, \ldots, d_{i_k} \rangle$ are the top $k$ smallest distances among $\langle d_1, \ldots, d_n \rangle$. After this, $C_2$ sends $\delta$ to $C_1$. Upon receiving $\delta$, $C_1$ proceeds as follows:

- Select the encrypted records $E_{pk}(t_{i_1}), \ldots, E_{pk}(t_{i_k})$ as the $k$-nearest records to $Q$ and randomize them attribute-wise. More specifically, $C_1$ computes $E_{pk}(\gamma_{j,h}) = E_{pk}(t_{i_j,h}) * E_{pk}(r_{j,h})$, for $1 \leq j \leq k$ and $1 \leq h \leq m$. Here $r_{j,h}$ is a random number in $Z_N$ and $t_{i_j,h}$ denotes the column $h$ attribute value of data record $t_{i_j}$. Send $\gamma_{j,h}$ to $C_2$ and $r_{j,h}$ to Bob, for $1 \leq j \leq k$ and $1 \leq h \leq m$.

Upon receiving $\gamma_{j,h}$, for $1 \leq j \leq k$ and $1 \leq h \leq m$, $C_2$ decrypts it to get $\gamma'_{j,h} = D_{sk}(\gamma_{j,h})$ and sends them to Bob. Note that, due to randomization by $C_1$, $\gamma'_{j,h}$ is always a random number in $Z_N$.

Finally, upon receiving $r_{j,h}$ from $C_1$ and $\gamma'_{j,h}$ from $C_2$, Bob computes the attribute values of $j$-th nearest neighbor to $Q$ as $t'_{j,h} = \gamma'_{j,h} - r_{j,h} \mod N$, for $1 \leq j \leq k$ and $1 \leq h \leq m$.

Algorithm 5 SkNN$_b(E_{pk}(Q), Q) \rightarrow \langle t'_{1}, \ldots, t'_{k} \rangle$

Require: $C_1$ has $E_{pk}(T)$; $C_2$ has $sk$; Bob has $Q$

1: Bob:
   (a) Compute $E_{pk}(q_j)$, for $1 \leq j \leq m$
   (b) Send $E_{pk}(Q) = \langle E_{pk}(q_1), \ldots, E_{pk}(q_m) \rangle$ to $C_1$

2: $C_1$ and $C_2$:
   (a) $C_1$ receives $E_{pk}(Q)$ from Bob
   (b) for $i = 1$ to $n$ do:
      - $E_{pk}(d_i) \leftarrow$ SSED$(E_{pk}(Q), E_{pk}(t_i))$
   (c) Send $\{\langle 1, E_{pk}(d_1) \rangle, \ldots, \langle n, E_{pk}(d_n) \rangle\}$ to $C_2$

3: $C_2$:
   (a) Receive $\{\langle 1, E_{pk}(d_1) \rangle, \ldots, \langle n, E_{pk}(d_n) \rangle\}$ from $C_1$
   (b) $d_i \leftarrow D_{sk}(E_{pk}(d_i))$, for $1 \leq i \leq n$
   (c) Generate $\delta \leftarrow \langle i_1, \ldots, i_k \rangle$, such that $(d_{i_1}, \ldots, d_{i_k})$
   (d) Send $\delta$ to $C_1$

4: $C_1$:
   (a) Receive $\delta$ from $C_2$
   (b) for $1 \leq j \leq k$ and $1 \leq h \leq m$ do:
      - $\gamma_{j,h} \leftarrow E_{pk}(t_{i_j,h}) * E_{pk}(r_{j,h})$, where $r_{j,h} \in R$ $Z_N$
      - Send $\gamma_{j,h}$ to $C_2$ and $r_{j,h}$ to Bob

5: $C_2$:
   (a) for $1 \leq j \leq k$ and $1 \leq h \leq m$ do:
      - Receive $\gamma_{j,h}$ from $C_1$
      - $\gamma'_{j,h} \leftarrow D_{sk}(\gamma_{j,h})$; send $\gamma'_{j,h}$ to Bob

6: Bob:
   (a) for $1 \leq j \leq k$ and $1 \leq h \leq m$ do:
      - Receive $r_{j,h}$ from $C_1$ and $\gamma'_{j,h}$ from $C_2$
      - $t'_{j,h} \leftarrow \gamma'_{j,h} - r_{j,h} \mod N$

B. Fully Secure kNN Protocol

The above-mentioned SkNN$_b$ protocol reveals the data access patterns to $C_1$ and $C_2$. That is, for any given $Q$, $C_1$ and $C_2$ know which data records correspond to the $k$-nearest neighbors of $Q$. Also, it reveals $d_i$ values to $C_2$. However, leakage of such information may not be acceptable in privacy-sensitive applications such as medical data. Along this direction, we propose a fully secure protocol, denoted by SkNN$_m$(where $m$ stands for maximally secure), to retrieve the $k$-nearest neighbors of $Q$. The proposed SkNN$_m$ protocol preserves all the desirable properties of a secure $k$NN protocol as mentioned in Section I.

The main steps involved in the proposed SkNN$_m$ protocol are as shown in Algorithm 6. Initially, Bob sends his attribute-wise encrypted query $Q$, that is, $E_{pk}(Q) = \langle E_{pk}(q_1), \ldots, E_{pk}(q_m) \rangle$ to $C_1$. Upon receiving, $C_1$ with private input $(E_{pk}(Q), E_{pk}(t_i))$ and $C_2$ with the secret key $sk$ jointly involve in the SSED protocol. The output of this step is $E_{pk}(d_i) = E_{pk}(|Q - t_i|^2)$ which will be known only to $C_1$, for $1 \leq i \leq n$. Then, $C_1$ with input $E_{pk}(d_i)$ and $C_2$ with $sk$ securely compute the encryptions of the individual bits of $d_i$ using the SBD protocol. Note that the output of this step $[d_i] = \langle E_{pk}(d_{i,1}), \ldots, E_{pk}(d_{i,i}) \rangle$ is known only to $C_1$, $C_2$, and $sk$.
Algorithm 6 SkNN$_m(E_{pk}(T), Q) → \langle t_1', \ldots, t_k' \rangle$

Require: C has $E_{pk}(T)$ and $\pi$; C$_2$ has $sk$; Bob has $Q$
1: Bob sends $E_{pk}(Q) = \langle E_{pk}(q_1), \ldots, E_{pk}(q_m) \rangle$ to C$_1$
2: C$_1$ and C$_2$:
   (a). C$_1$ receives $E_{pk}(Q)$ from Bob
   (b). for $i = 1$ to $n$ do:
       • $E_{pk}(d_i) ← SSED(E_{pk}(Q), E_{pk}(t_i))$
       • $[d_i] ← SBD(E_{pk}(d_i))$
3: for $s = 1$ to $k$ do:
   (a). C$_1$ and C$_2$:
       • $[d_{min}] ← \text{SMIN}([d_1], \ldots, [d_n])$
   (b). C$_1$:
       • $E_{pk}(d_{min}) ← \prod_{\gamma=0}^{l-1} E_{pk}(d_{min,\gamma+1})2^{2^\gamma-1}$
       • if $s \neq 1$ then, for $1 \leq i \leq n$ do:
           • $E_{pk}(d_i) ← \prod_{\gamma=0}^{l-1} E_{pk}(d_i,\gamma+1)2^{2^\gamma-1}$
       • for $i = 1$ to $n$ do:
           • $\tau_i ← E_{pk}(d_{min}) \ast E_{pk}(d_i)^{N-1}$
           • $\tau_i' ← \tau_i'$, where $r_i \in R Z_N$
           • $\beta ← \pi(\tau');$ send $\beta$ to C$_2$
   (c). C$_2$:
       • Receive $\beta$ from C$_1$
       • $\beta_i' ← D_{sk}(\beta_i)$, for $1 \leq i \leq n$
       • Compute $U$, for $1 \leq i \leq n$:
           • if $\beta_i' = 0$ then $U_i = E_{pk}(1)$
           • else $U_i = E_{pk}(0)$
     • Send $U$ to C$_1$
   (d). C$_1$:
       • Receive $U$ from C$_2$ and compute $V ← \pi^{-1}(U)$
       • $V_{i,j} ← \text{SM}(V_i, E_{pk}(t_{i,j}))$, for $1 \leq i \leq n$ and $1 \leq j \leq m$
       • $E_{pk}(t_{\gamma,j}) ← \prod_{\gamma=1}^{\gamma} V_{i,j}$, for $1 \leq i \leq n$ and $1 \leq j \leq m$
       • $E_{pk}(t_\gamma) = \langle E_{pk}(t_{1,1}), \ldots, E_{pk}(t_{l,m}) \rangle$
   (e). C$_1$ and C$_2$, for $1 \leq i \leq n$:
       • $E_{pk}(d_{i,\gamma}) ← \text{SBOR}(V_i, E_{pk}(d_{i,\gamma}))$, for $1 \leq \gamma \leq l$

The rest of the steps are similar to steps 4-6 of SkNN$_b$

where $d_{i,1}$ and $d_{i,i}$ are the most and least significant bits of $d_i$ respectively. Note that $0 \leq d_i < 2^l$, for $1 \leq i \leq n$.

After this, C$_1$ and C$_2$ compute the top $k$ (in encrypted form) records that are closest to $Q$ in an iterative manner. More specifically, they compute $E_{pk}(t_1')$ in the first iteration, $E_{pk}(t_2')$ in the second iteration, and so on. Here $t_1'$ denotes the $k$th nearest neighbor to $Q$, for $1 \leq s \leq k$. At the end of $k$ iterations, only C$_1$ knows $(E_{pk}(t_1'), \ldots, E_{pk}(t_k'))$. To start with, in the first iteration, C$_1$ and C$_2$ jointly compute the encryptions of the minimum individual bits as below:

\[
E_{pk}(d_{\min}) = \prod_{\gamma=0}^{l-1} E_{pk}(d_{\min,\gamma+1})2^{2^\gamma-1}
\]

where $d_{\min,1}$ and $d_{\min,i}$ are the most and least significant bits of $d_{\min}$ respectively.

• Compute the encryption of $d_{\min}$ from its encrypted individual bits as below

\[
E_{pk}(d_{\min}) = \prod_{\gamma=0}^{l-1} E_{pk}(d_{\min,\gamma+1})2^{2^\gamma-1} + \cdots + d_{\min,i}
\]

where $d_{\min,1}$ and $d_{\min,i}$ are the most and least significant bits of $d_{\min}$ respectively.

• Compute the encryption of difference between $d_{\min}$ and each $d_i$. That is, C$_1$ computes $\tau_i = E_{pk}(d_{\min}) \ast E_{pk}(d_i)^{N-1} = E_{pk}(d_{\min} - d_i)$, for $1 \leq i \leq n$.

• Randomize $\tau_i$ to get $\tau_i' = \tau_i' = E_{pk}(\tau_i \ast (d_{\min} - d_i))$, where $\tau_i$ is a random number in $Z_N$. Note that $\tau_i'$ is an encryption of either 0 or a random number, for $1 \leq i \leq n$.

Also, permute $\tau'$ using a random permutation function $\pi$ (known only to C$_1$) to get $\beta = \pi(\tau')$ and send it to C$_2$.

Upon receiving $\beta$, C$_2$ decrypts it component-wise to get $\beta_i' = D_{sk}(\beta_i)$, for $1 \leq i \leq n$. After this, he/she computes an encrypted vector $U$ of length $n$ such that $U_i = E_{pk}(1)$ if $\beta_i' = 0$, and $E_{pk}(0)$ otherwise. Here we assume that exactly one of the entries in $\beta$ equals to zero and rest of them are random. This further implies that exactly one of the entries in $U$ is an encryption of 1 and rest of them are encryptions of 0’s. However, we emphasize that if $\beta'$ has more than one 0’s, then C$_2$ can randomly pick one of those indexes and assign $E_{pk}(1)$ to the corresponding index of $U$ and $E_{pk}(0)$ to the rest. Then, C$_2$ sends $U$ to C$_1$. After receiving $U$, C$_1$ performs inverse permutation on it to get $V = \pi^{-1}(U)$. Note that exactly one of the entry in $V$ is $E_{pk}(1)$ and the remaining are encryption of 0’s. In addition, if $V_i = E_{pk}(1)$, then $t_i$ is the closest record to $Q$. However, C$_1$ and C$_2$ do not know which entry in $V$ corresponds to $E_{pk}(1)$.

Finally, C$_1$ computes $E_{pk}(t_1')$, encryption of the closest record to $Q$, and updates the distance vectors as follows:

• C$_1$ and C$_2$ jointly involve in the secure multiplication (SM) protocol to compute $V_{i,j}' = V_i \ast E_{pk}(t_{i,j})$, for $1 \leq i \leq n$ and $1 \leq j \leq m$. The output $V'$ from the SM protocol is known only to C$_1$. After this, by using homomorphic properties, C$_1$ computes the encrypted record $E_{pk}(t_1') = \langle E_{pk}(t_{1,1}), \ldots, E_{pk}(t_{1,m}) \rangle$ locally, $E_{pk}(t_1') = \prod_{i=1}^{m} V_{i,j}'$, where $1 \leq j \leq m$. Note that $t_{1,j}$ denotes the $j$th attribute value of record $t_1'$.

It is important to note that the first nearest tuple to $Q$ should be obliviously excluded from further computations. However, since C$_1$ does not know the record corresponding to $E_{pk}(t_1')$, we need to obliviously eliminate the possibility of choosing this record again in next iterations. For this, C$_1$ obliviously updates the distance corresponding to $E_{pk}(t_1')$ to the maximum value, i.e., $2^l - 1$. More specifically, C$_1$ updates the distance vectors with the help of C$_2$ using the SBOR protocol as $E_{pk}(d_{i,\gamma}) = \text{SBOR}(V_i, E_{pk}(d_{i,\gamma}))$, for $1 \leq i \leq n$ and $1 \leq \gamma \leq l$. Note that when $V_i = E_{pk}(1)$, the corresponding distance vector $d_i$ is set to the maximum value. That is, under this case, $[d_i] = \langle E_{pk}(1), \ldots, E_{pk}(1) \rangle$. However, when $V_i = E_{pk}(0)$, the OR operation has no affect on $d_i$. 
The above process is repeated until \(k\) iterations, and in each iteration \(d_{t_i}\) corresponding to the current chosen record is set to the maximum value. However, since \(C_1\) does not know which \(d_{t_i}\) is updated, he/she has to re-compute \(E_{pk}(d_{t_i})\) in each iteration using the corresponding \(d_{t_i}\), for \(1 \leq i \leq n\). In iteration \(s\), \(E_{pk}(t'_s)\) is known only to \(C_1\).

At the end of the iterative step (i.e., step 3 of Algorithm 6), \(C_1\) has \(\langle E_{pk}(t'_1), \ldots, E_{pk}(t'_k) \rangle\) - the list of encrypted records of \(k\)-nearest neighbors to the input query \(Q\). The rest of the process is similar to steps 4 to 6 of Algorithm 5. Briefly, \(C_1\) randomizes \(E_{pk}(t'_j)\) attribute-wise to get \(\gamma_j,h = E_{pk}(t'_j,h) \ast E_{pk}(r_{j,h})\) and sends \(\gamma_j,h\) to \(C_2\) and \(r_{j,h}\) to \(Bob\), for \(1 \leq j \leq k\) and \(1 \leq h \leq m\). Here \(r_{j,h}\) is a random number in \(\mathbb{Z}_N\). Upon receiving \(\gamma_j,h\)'s, \(C_2\) decrypts them to get the randomized \(k\)-nearest records as \(\gamma'_j,h = D_{sk}(\gamma_j,h)\) and sends them to \(Bob\), for \(1 \leq j \leq k\) and \(1 \leq h \leq m\). Finally, upon receiving \(r_{j,h}\) from \(C_1\) and \(\gamma'_j,h\) from \(C_2\), \(Bob\) computes the \(j^{th}\) nearest neighboring record to \(Q\), as \(t_{j,h}' = \gamma'_j,h - r_{j,h} \mod N\), for \(1 \leq j \leq k\) and \(1 \leq h \leq m\).

C. Security Analysis

First, due to the encryption of \(Q\) and by semantic security of the Paillier cryptosystem, \(Bob\)'s input query \(Q\) is protected from \(Alice\), \(C_1\) and \(C_2\) in both protocols.

In the \(SkNN_b\) protocol, the decryption operations at step 3(b) of Algorithm 5 reveal \(d_i\) values to \(C_2\). In addition, since \(C_2\) generates the top \(k\) index list (at step 3(c) of Algorithm 5) and sends it to \(C_1\), the data access patterns are revealed to \(C_1\) and \(C_2\). Therefore, our basic \(SkNN_b\) protocol is secure under the assumption that \(d_i\) values can be revealed to \(C_2\) and data access patterns can be revealed to \(C_1\) and \(C_2\).

On the other hand, the security analysis of \(SkNN_m\) is as follows. At step 2 of Algorithm 6, the outputs of SSED and SBD are in encrypted format, and are known only to \(C_1\). In addition, all the intermediate results decrypted by \(C_2\) in SSED are uniformly random in \(\mathbb{Z}_N\). Also, as mentioned in [23], the SBD protocol is secure. Thus, no information is revealed during step 2 of Algorithm 6. In each iteration, the output of \(SMIN_n\) is known only to \(C_1\) and no information is revealed to \(C_2\). Also, \(C_1\) and \(C_2\) do not know which record belongs to current global minimum. Thus, data access patterns are protected from both \(C_1\) and \(C_2\). At step 3(c) of Algorithm 6, a component-wise decryption of \(\beta\) reveals the tuples that satisfy the current global minimum distance to \(C_2\). However, due to permutation by \(C_1\), \(C_2\) cannot trace back to the corresponding data records. Also, note that decryption of \(\beta\) gives either encryptions of 0’s or random numbers in \(\mathbb{Z}_N\). Similarly, since \(U\) is an encrypted vector, \(C_1\) cannot know which tuple corresponds to current global minimum distance. Thus, data access patterns are further protected at this step from \(C_1\). In addition, the update process at step 3(e) of Algorithm 6 does not leak any information to \(C_1\) and \(C_2\). In summary, \(C_1\) and \(C_2\) do not know which data records correspond to the output set \(\langle t'_1, \ldots, t'_k \rangle\).

Based on the above discussions, it is clear that the proposed \(SkNN_m\) protocol protects the confidentiality of the data, privacy of user’s input query, and hides the data access patterns.

D. Complexity Analysis

The computation complexity of \(SkNN_b\) is bounded by \(O(n \ast m + k)\) encryptions, decryptions and exponentiations. In practice \(k \ll n \ast m\); therefore, the computation complexity of \(SkNN_b\) is bounded by \(O(n \ast m)\) encryptions and exponentiations (assuming that encryption and decryption operations under Paillier cryptosystem take similar amount of time). On the other hand, the computation complexity of \(SkNN_m\) is bounded by \(O(n \ast (l + m + k \ast \log_2 n))\) encryptions and exponentiations. Due to space limitations, we refer the reader to our technical report [25] for a detailed complexity analyses.

Depending on the encryption key size, the overall computation cost of the proposed \(SkNN_m\) (more expensive than \(SkNN_b\)) is between 2 and 3 orders of magnitude higher than the non-crypto cases (the related works acknowledged in the paper). This is the cost we need to pay to maximize data confidentiality. However, on the user or client side, the running time is comparable to the non-crypto case since the user only performs a very small number of encryption operations (bounded by the number of attributes) which was done in less than a second as shown in our experiments. Our goal is to outsource all or most computations to the cloud so that the user can issue queries using any mobile device with limited storage and computing capability. Note that, data confidentiality is fully protected under the proposed \(SkNN_m\) protocol.

V. Empirical Results

In this section, we discuss the performances of the proposed protocols in detail under different parameter settings. We used Paillier cryptosystem [15] and implemented the proposed protocols in C. Various experiments were conducted on a Linux machine with an Intel® Xeon® Six-Core⃝ CPU 3.07 GHz processor and 12GB RAM running Ubuntu 10.04 LTS.

Since it is difficult to control the parameters in a real dataset, we randomly generated synthetic datasets depending on the parameter values in consideration. Using these synthetic datasets we can perform a more elaborated analysis on the computation costs of the proposed protocols under different parameter settings. We encrypted these datasets attribute-wise, using the Paillier encryption whose key size is varied in our experiments, and the encrypted data were stored on our machine. Based on the proposed protocols, we then executed a random query over this encrypted data. For the rest of this section, we do not discuss about the performance of Alice since it is a one-time cost. Instead, we evaluate and analyze the performances of \(SkNN_b\) and \(SkNN_m\) separately. In addition, we compare the two protocols. In our experiments, the Paillier encryption key size \(K\) is set to either 512 or 1024 bits.

A. Performance of \(SkNN_b\)

In this sub-section, we analyze the computation costs of \(SkNN_b\) by varying the number of data records \((n)\), number of attributes \((m)\), number of nearest neighbors \((k)\), and encryption
Running time of S depends on (or grows linearly with) \( n \) respectively. This further shows that our protocols are very efficient from user’s perspective. In S, SSED is the bottleneck whereas in S the bottleneck is SMIN.

B. Performance of S

We also evaluated the computation costs of S for varying values of \( k, l \) and \( K \). Throughout this sub-section, we fix \( m = 6 \) and \( n = 2000 \). However, we observed that the running time of S grows almost linearly with \( n \) and \( m \).

For \( K = 512 \) bits, the computation costs of S for varying \( k \) and \( l \) are as shown in Figure 2(d). Following from Figure 2(d), for \( l = 6 \), the running time of S varies from 11.93 to 55.65 minutes when \( k \) is changed from 5 to 25 respectively. Also, for \( l = 12 \), the running time of S varies from 20.68 to 97.8 minutes when \( k \) is changed from 5 to 25 respectively. In either case, the cost of S grows almost linearly with \( k \) and \( l \).

A similar trend is observed for \( K = 1024 \) as shown in Figure 2(e). In particular, for any given fixed parameters, we identified that the computation cost of S increases almost a factor of 7 when \( K \) is doubled. For example, when \( k = 10 \), S took 22.85 and 157.17 minutes to generate the 10 nearest neighbors of \( Q \) under \( K = 512 \) and 1024 respectively. Furthermore, when \( k = 5 \), we observed that around 69.7% of cost in S is accounted due to SMIN, which is initiated \( k \) times in S (once in each iteration). Also, the cost incurred due to SMIN increases from 69.7% to at least 75% when \( k \) is increased from 5 to 25.

In addition, by fixing \( n = 2000, m = 6, l = 6 \) and \( K = 512 \), we compared the running times of both protocols for varying values of \( k \). As shown in Figure 2(f), the running time of S varies from 11.93 to 55.65 minutes as we increase \( k \) from 5 to 25.

Based on the above results, it is clear that the computation costs of S are significantly higher than that of S. However, we emphasize that S is more secure than S; therefore, the two protocols act as a trade-off between security and efficiency. Also, it is important to note that Bob’s computation cost is mainly due to the encryption of his input query record. As an example, for \( m = 6 \), Bob’s computation costs are 4 and 17 milliseconds when \( K = 512 \) and 1024 respectively. This further shows that our protocols are very efficient from end-user’s perspective. In S, SSED is the bottleneck whereas in S the bottleneck is SMIN.
C. Towards Performance Improvement

At first, it seems that the proposed protocols are costly and may not scale well for large datasets. However, in both protocols, we emphasize that the computations involved on data records for efficiency purpose. To further justify this claim, we implemented a parallel version of our S\(k\)NN\(_b\) protocol using OpenMP programming and compared its computation costs with its serial version. As mentioned earlier, our machine has 6 cores which can be used to perform parallel operations on 6 threads. For \(m = 6, k = 5\) and \(K = 512\) bits, the running time of serial and parallel versions on S\(k\)NN\(_b\) are 40 and 215.59 seconds respectively.

We believe that similar efficiency gains can be achieved by parallelizing the operations on S\(k\)NN\(_m\). Based on the above discussions, especially in a cloud computing environment where high performance parallel processing can easily be achieved, we claim that the scalability issue of the proposed protocols can be eliminated or mitigated. In addition, using the existing map-reduce techniques, we can drastically improve the performance further by executing parallel operations on multiple nodes. We will leave this analysis to future work.

VI. Conclusion

The \(k\)-nearest neighbors is one of the commonly used query in many data mining applications. Under an outsourced database environment, where encrypted data are stored in the cloud, secure query processing over encrypted data becomes challenging. The existing S\(k\)NN techniques over encrypted data are not secure. In this paper, we proposed two novel S\(k\)NN protocols over encrypted data in the cloud. The first protocol, which acts as a basic solution, leaks some information to the cloud. On the other hand, our second protocol is fully secure, that is, it protects the confidentiality of the data, user’s input query, and also hides the data access patterns. However, the second protocol is more expensive compared to the basic protocol. Also, we evaluated the performance of our protocols under different parameter settings. As a future work, we will investigate and extend our research to other complex conjunctive queries over encrypted data.

Acknowledgments

The authors would like to thank the anonymous reviewers for their helpful comments. This material is based upon work supported by the Office of Naval Research under Award No. N000141110256 and NSF under award No. CNS-1011984.

References