Secure Nearest Neighbor Revisited

Bin Yao[1], Feifei Li[2], Xiaokui Xiao[3]

[1]*Department of Computer Science and Engineering, Shanghai Key Laboratory of Scalable Computing and Systems, Shanghai Jiao Tong University, China*

[2]*School of Computing, University of Utah*

[3]*School of Computer Engineering, Nanyang Technological University, Singapore*

[1]{yaobin}@cs.sjtu.edu.cn,[2]{lifeifei}@cs.utah.edu,[3]{xkxiao}@ntu.edu.sg

***Abstract*—In this paper, we investigate the secure nearest neighbor (SNN) problem, in which a client issues an encrypted query point $E(q)$ to a cloud service provider and asks for an encrypted data point in $E(D)$ (the encrypted database) that is closest to the query point, without allowing the server to learn the plaintexts of the data or the query (and its result). We show that efficient attacks exist for existing SNN methods [15], [21], even though they were claimed to be secure in standard security models (such as indistinguishability under chosen plaintext or ciphertext attacks). We also establish a relationship between the SNN problem and the order-preserving encryption (OPE) problem from the cryptography field [5], [6], and we show that SNN is at least as hard as OPE. Since it is impossible to construct secure OPE schemes in standard security models [5], [6], our results imply that one cannot expect to find the exact (encrypted) nearest neighbor based on only $E(q)$ and $E(D)$. Given this hardness result, we design new SNN methods by asking the server, given only $E(q)$ and $E(D)$, to return a relevant (encrypted) partition $E(G)$ from $E(D)$ (i.e., $G \subseteq D$), such that that $E(G)$ is guaranteed to contain the answer for the SNN query. Our methods provide customizable tradeoff between efficiency and communication cost, and they are as secure as the encryption scheme E used to encrypt the query and the database, where E can be any well-established encryption schemes.**

I. Introduction

The cloud has gained increasing popularity for its flexibility and scalability, which motivates cloud service providers to offer accesses to cloud databases, such as Amazon Relational Database Service (Amazon RDS), Google Cloud SQL, and Microsoft SQL Azure. Data owners outsource their databases to the cloud service providers and rely on their services for storage, management, and query processing of the databases. Clearly, this framework offers great flexibility and scalability to data owners and their clients, and it is especially useful for users with stringent local resources.

However, the remote placement of the data brings security concerns. A data owner may prefer to prevent the server (i.e., the service provider) from learning the content of his/her database D or the contents of queries to D, while still requiring the server to provide database functionality for D in the cloud [11], [13], [16]. For this purpose, the data owner needs to encrypt D with an encryption scheme E and only publishes to the server an encrypted version of D, denoted as $E(D)$. The clients also need to encrypt their queries q and send only $E(q)$ to the server. The server needs to identify the ciphertext in $E(D)$ that corresponds to the answer of q on D, using only $E(q)$ and $E(D)$. We dub a query processing procedure satisfying these constraints as *secure query processing*.

Under the settings of secure query processing, there have been a few techniques that address general range selection queries and aggregation queries [11], [13], [14], [16], [19]. For more complicated query types, however, there is only a narrow selection of existing studies. In particular, this has been the case for the *secure nearest neighbor (SNN)* problem, where a client issues an encrypted query point $E(q)$ to a server and asks for an encrypted data point in $E(D)$ that is closest to the query point, without allowing the server to learn the plaintext of the query or the data. This problem is of significant practical importance, since nearest neighbor (NN) queries are fundamental in spatial and multimedia databases, both of which find extensive applications in the cloud. Unfortunately, as we will show in this work, existing methods for secure NN queries, though claimed to be secure [15], [21] in standard security models (e.g., indistinguishability under chosen plaintext or ciphertext attacks), *are not secure*.

This inspires us to examine the hardness of the SNN problem. Our investigation shows that the SNN problem is at least as hard as constructing an order-preserving encryption (OPE) scheme [5], [6]. Given the recent findings that it is impossible to construct secure OPE schemes in standard security models [5], our results immediately imply that one cannot expect the server to find the exact (encrypted) answer for an SNN query given only $E(q)$ and $E(D)$.

However, this does not mean we are hopeless in answering secure NN queries in cloud databases. The trick is to relax what we require the server to achieve. Instead of asking him to find the encrypted exact nearest neighbor based on only $E(q)$ and $E(D)$, we can ask the server to find a relevant (encrypted) partition $E(G)$ from $E(D)$ (i.e., $G \subseteq D$), such that G is guaranteed to contain the nearest neighbor of q. Note that the hardness of the SNN problem does not apply to this modified problem, as finding $E(G)$ is not the same as finding the encrypted exact nearest neighbor. Once given $E(G)$, any trusted entity with the decryption function E^{-1} can easily identify the answer to a query q, by using E^{-1} to decrypt $E(G)$ and then inspecting the data points therein.

Our new methods build on this idea for data in one and two dimensions, and they allow customizable tradeoff between communication and computation costs in cloud databases. Our idea leverages on special partitions over D and the Voronoi diagram of D. We dub our new method the *secure Voronoi diagram (*SVD*)* method. Since the SVD method does not use any new encryption schemes, rather, it only relies on any

standard encryption scheme E (e.g., public-key encryption RSA, symmetric-key encryption AES), the SVD method is as secure as E for any standard security model in which E is proven secure (e.g., indistinguishability in either chosen plaintext or chosen ciphertext attacks).

Finally, we use extensive experiments to demonstrate the efficiency of our attacks to existing methods, as well as the efficiency and the scalability of our new method SVD. Our experiments were conducted over real and large datasets (up to few millions points in multi-dimensional spaces).

The rest of the paper is organized as follows. Section II formulates the problem. Sections III-A and III-B show our construction of two (different) efficient attacks to the SNN schemes in [21] and [15] respectively. Section IV investigates the relationship between SNN and OPE, which establish the hardness of the SNN problem. Section V presents the new, secure Voronoi diagram (SVD) method. Section VI gives our experimental results. Section VII surveys the related work, and Section VIII concludes the paper.

II. Problem Formulation

Formally, the SNN problem involves three parties:

1) A *data owner* who has a database D with d-dimensional Euclidean objects/points, and would like to outsource D to a server that cannot be fully trusted.
2) A *client* (or multiple of them) who wants to access and pose queries to the database D.
3) A *server* that is *honest but potentially curious* in the tuples in D and/or the queries from the clients. A sever could be curious either because he is just curious or he has been compromised to become curious on the behalf of a third party without his explicit knowledge.

Our objective is to enable the client to perform NN queries without letting the server learn contents about the query (and its result) or the tuples in the database. Note that in practice, a data owner could also be a client. Clearly, in order to achieve our objective, the database D has to be encrypted by some encryption scheme E by the data owner. We use $E(D)$ to denote the encrypted version of the database, and E^{-1} to denote the corresponding decryption function (with the necessary secret). Similarly to the case for the data owner, clients only send to the server the encrypted versions $E(q)$ of their queries q (each of which represents a query point).

We aim to ensure that the SNN method is as secure as the encryption method E used by the data owner. For ease of exposition, we consider that E is proven secure under chosen-plaintex-attack, but our method can be straightforwardly extended for other adversary models (e.g., indistinguishability under chosen-ciphertext-attack).

The above formulation of the SNN problem is adopted in previous work [15], [21] and is of considerable practical importance for various reasons [11], [12], [14]–[16], [19], [21]. For example, D might contain some sensitive values that cannot be disclosed to the server, or D represents a business asset to the data owner and/or the client, or the service providers wish to provide SNN service as an assurance to attract more customers.

Without loss of generality, we assume that D is represented by N tuples $\{p_1, \ldots, p_N\}$. Each tuple can be viewed as a d-dimensional point: for any $p \in D$, $p = \langle v_1, v_2, \ldots, v_d \rangle$; p can also be considered as a vector in d-dimensional space.

Remarks. One may consider an alternative formulation of the SNN problem by regarding the data owner and the client as the same party, since they share the same key and are assumed to trust each other. We explicitly differentiate the data owner from the client, so as to be consistent with previous work [15], [21] and to simplify our discussion on the loopholes of the existing methods (see Section III).

Unless otherwise specified, all proofs in this paper can be found in Appendix A from our online technical report [22].

III. Insecurity of Existing Methods

A. Solution by Wong et al. [21]

Basic idea. The data owner encrypts each tuple in the database before sending it to the server using a customized encryption scheme E_T. The client sends an encrypted query using a related but slightly different customized encryption scheme E_Q to the server. Wong et al. [21] showed that the customized encryption schemes E_T and E_Q preserve the dot product between the query vector q and any tuple vector p from the database D, i.e., $q \cdot p = E_Q(q) \cdot E_T(p)$. Furthermore, for any two points p_1 and p_2 from D, $p_1 \cdot p_2 \neq E_T(p_1) \cdot E_T(p_2)$, i.e, the dot product between two tuples in D cannot be obtained directly from their ciphertext.

Given an encrypted query point $E(q)$, the server (i) inspects the dot product between $E(q)$ and the ciphertext $E_T(p)$ of each tuple $p \in D$, and (ii) returns to the client the ciphertext that corresponds to the answer of the SNN query (see [21] for the detailed algorithm). After that, the client decrypts the ciphertext to get the query result. Note that, in order for the client to encrypt and decrypt, the data owner needs to share the secret key with the client.

Customized encryption schemes. The data owner and the client share the following secret information: (i) an integer $d' \geq d$, (ii) two $d' \times d'$ matrices M_1 and M_2 that are random but invertible, (iii) a random bit vector A with d' bits b_1, b_2, …, $b_{d'}$, and (iv) some additional information that is irrelevant to our discussion.

Let $p = \langle v_1, v_2, \ldots, v_d \rangle$ be any tuple in the database. The encryption scheme E_T works as follows. The data owner first converts p into a d'-dimensional tuple $p' = [v'_1, ..., v'_{d'}]^T$, such that (i) $v'_i = v_i$ for any $i <= d$, (ii) $v_{d+1} = -\frac{1}{2}\sum_{j=1}^{d} v_j^2$, and (iii) v'_{d+2}, ..., $v'_{d'}$ are random numbers generated by some specific rules (omitting such details does not affect our analysis here). Next, p' is transformed into two tuples $p'_a = [x_1, \ldots, x_{d'}]^T$ and $p'_b = [y_1, \ldots, y_{d'}]^T$ based on A, such that for any $i \in [1, d']$,

1) if $b_i = 1$ (the ith bit from A), then x_i and y_i are two random numbers satisfying $x_i + y_i = v'_i$;
2) otherwise, $x_i = y_i = v'_i$.

At last, the data owner computes $p_a^* = M_1^T \cdot p'_a$ and $p_b^* = M_2^T \cdot p'_b$, and sends p_a^* and p_b^* to the server as the encrypted version of p.

Meanwhile, the scheme E_Q for encrypting query points is as follows. Given $q = \langle v_1, v_2, \ldots, v_d \rangle$ and a random number $r > 0$, the client first converts q into a d'-dimensional tuple $q' = [v'_1, ..., v'_{d'}]^T$ that satisfies two conditions. First, $v'_i = rv_i$ for any $i <= d$, and $v'_{d+1} = r$. Second, v'_i's for $i \in [d+2, d']$ are random values generated according to some specific rules, such that *the scalar product over the artificial attributes (between $d+1$ and d')* from any q' and p' is always 0.

Then q' is transformed into two tuples $q'_a = [x_1, \ldots, x_{d'}]^T$ and $q'_b = [y_1, \ldots, y_{d'}]^T$, such that $\forall i \in [1, d']$,

1) if $b_i = 0$ (the ith bit from A), then x_i and y_i are two random numbers such that $x_i + y_i = v'_i$;
2) otherwise, $x_i = y_i = v'_i$.

The encrypted version of q is the pair: $q_a^* = M_1^{-1} \cdot q'_a$ and $q_b^* = M_2^{-1} \cdot p'_b$.

Security guarantee. Wong et al. show that the above encryption schemes preserve the dot product between any query point q and any point p in the database, i.e., $q \cdot p = E_T(q) \cdot E_Q(p)$ (this forms the basis for their SNN method [21]). Furthermore, Wong et al. claim that this protocol can guard against any attacks based on the knowledge of a number of (plaintext, ciphertext) pairs [21], and their argument is as follows: if the boolean vector A is known to the adversary, then he/she would be able to use the known (plaintext, ciphertext) pairs to construct linear equations about M_1 and M_2, and then solve the equations to obtain M_1 and M_2. However, since A is secret, it would be hard for the adversary to derive the correct linear equations to use, since the formulation of the equations depends on A. A brute-force approach would require the adversary to examine all possible bit vector A, which leads to $2^{|A|}$ linear equation systems that cannot be solved in reasonable time when $|A|$ is large.

We observe that the above reasoning is not rigorous, since it assumes that the adversary only attempts his/her attacks by solving M_1 and M_2. We will demonstrate an attack that does not require any knowledge about M_1 and M_2.

Our attack using chosen-plaintext attack. Assume that the server obtains d query points and their encryption (by asking the oracle in the chosen plaintext attack model). For each q of those query points, the server would have two encrypted points q_a^* and q_b^* generated by E_Q. Wong et al.'s encryption schemes ensure that the dot product between q and any database point p can be calculated based on the following equation:

$$p \cdot q = p_a^* \cdot q_a^* + p_b^* \cdot q_b^*, \quad (1)$$

Notice that the above equation contains only d variables unknown to the server, i.e., the d coordinates of the data point p. Since the server has the plaintext-ciphertext pair of d query points, he can construct d linear equations like (1) to derive the coordinates of p.

B. Solution by Hu et al. [15]

Hu et al. [15] consider the SNN problem under a setting that is different but similar to ours. They assume that each client (i) has the ciphertexts of all data points in D and the encryption function E used to encrypt D, but (ii) does not have the decryption function E^{-1}. On the other hand, the server has E^{-1} and some auxiliary information about each data point (which is irrelevant to our discussion), but does not have the plaintext or ciphertext of any data point. The objective of Hu et al.'s method is to enable the client to identify the encrypted answer for any SNN query (with the help from the server), after which the client can retrieve the auxiliary information associated with the answer from the server (the plaintext of the answer remains secret to the client). Hu et al. claimed that their solution not only prevents the server and client to learn the plaintext of any data point, but also prevents the data owner and the server from learning the queries posed by the clients. However, we will show that Hu et al.'s solution does not fulfill their security claims.

The encryption scheme. The construction in [15] relies on an encryption scheme that gives what they called the privacy homomorphism (PH). PH is an encryption transformations which map some operations on cleartext to operations on ciphertext. Formally, they are encryptions $E_k : P \rightarrow X$ that allow a set of operations F on encrypted data without knowledge of the decryption function (here, P is the domain of plaintext and X is the domain of ciphertext). In particular, they used the ASM-PH encryption scheme from [9], which supports modular addition, subtraction, and multiplication. In what follows, we use E to denote an ASM-PH encryption function, and E^{-1} to denote the corresponding decryption function (with the necessary secret keys). The PH encryption E in [9] is shown to be secure against known-plaintext attacks, and it can perform addition, subtraction, and multiplication directly on the ciphertexts, e.g., $E(a+b) = E(a) + E(b)$. Note that in PH the operation on $E(a)$ and $E(b)$ is not necessarily the same as that in plaintexts to preserve an operation, but maybe some function f on $E(a)$ and $E(b)$ that is efficient to compute and gives the same output as the operation of interest over the corresponding plaintexts, i.e., $E(a+b) = f(E(a), E(b))$. But for simplicity, in the remainder of this paper, we just use the same operation over the ciphertexts to denote such an f.

However, assuming a fully secure ASM-PH scheme E, we can still launch an efficient attack on Hu et al.'s solution.

Basic idea of Hu et al.'s solution. The data owner builds a multidimensional index I over his database D. Each node b in I has a set of entries, where each entry e has a key value v and a pointer p (to a child node of b). Note that v can represent any object. For example, in the case of a d-dimensional R-tree, v is an minimum bounding rectangle (MBR) and can be represented as $(\vec{\ell}, \vec{u})$, where $\vec{\ell} = \{\ell_1, \ldots, \ell_d\}$ and $\vec{u} = \{u_1, \ldots, u_d\}$, such that $[\ell_i, u_i]$ is the extent of the MBR in the ith dimension. After I is generated, the data owner constructs a *shadow index* $E(I)$, which is identical to I except that the key values (and only the key values) in all entries from all nodes are encrypted by an ASM-PH encryption function E. $E(I)$ is published to all clients, and only the decryption function E^{-1} of E is sent to the server.

During query processing, Hu et al.'s solution requires the client to encrypt the query q and traverse the shadow index $E(I)$ locally with the help of the server. In particular, for each node b in $E(I)$ visited by the client and for each entry $e = (E(v), p)$ in b, the client computes $E(\text{mindist}(q, v)) = \text{mindist}(E(q), E(v))$ using the properties of the ASM-PH encryption E, where $\text{mindist}(\cdot)$ is the minimum distance between a query point q and an MBR v. Hu et al. showed that in the ith dimension (suppose that $q = (q_1, \ldots, q_d)$):

$$E(2 \cdot \text{mindist}(q_i, [\ell_i, u_i])) = sign(u_i - q_i)(E(u_i) - E(q_i)) + sign(\ell_i - q_i)(E(\ell_i) - E(q_i)) - (E(u_i) - E(\ell_i)), \text{ and} \quad (2)$$

$$\text{mindist}(E(q), E(v)) = \sum_{i=1}^{d} E^2(2 \cdot \text{mindist}(q_i, [\ell_i, u_i])), \quad (3)$$

where $sign(a - b)$ returns -1 if $a < b$ and 1 otherwise.

Note that in (2), only $sign(u_i - q_i)$ and $sign(\ell_i - q_i)$ are not known/computable by the client locally. To figure out their values, the client computes $E(u_i) - E(q_i) = E(u_i - q_i)$ and $E(\ell_i) - E(q_i) = E(\ell_i - q_i)$ locally and sends them to the server. The server, with the decryption function E^{-1}, can easily tell the values for $sign(u_i - q_i)$ and $sign(\ell_i - q_i)$ and send them back to the client, given $E(u_i - q_i)$ and $E(\ell_i - q_i)$. Now, knowing the values of $sign(u_i - q_i)$ and $sign(\ell_i - q_i)$, the client proceeds to compute $E(2 \cdot \text{mindist}(q_i, [\ell_i, u_i]))$ by (2) and then derive $\text{mindist}(E(q), E(v))$ by (3). Finally, the client sends $\text{mindist}(E(q), E(v))$, which is equivalent to $E(\text{mindist}(q, v))$, to the server for decryption to figure out the actual distance between q and the MBR represented by v. The client repeats this process for each entry $e = (E(v), p)$ from an index node u, and chooses the proper children nodes to browse in the next level, following the standard NN search algorithm in an R-tree.

To prevent the server from knowing the exact value of $u_i - q_i$ and $\ell_i - q_i$, before sending $E(u_i - q_i)$ and $E(\ell_i - q_i)$ to the server, the client subjects them to a *scrambling process*. Similarly, to prevent the client from knowing the exact value of $\text{mindist}(q, v)$. The server subjects the decrypted value of $\text{mindist}(E(q), E(v))$ to a *recoding process*.

The details of the *scrambling and recoding* procedures are not important to our discussion here. The bottom line is, in Hu et al.'s scheme, $sign(u_i - q_i)$ and $sign(\ell_i - q_i)$ must be computed by server and sent to client for each dimension $i \in [1, d]$, so that client can compute some distances locally to decide the next node(s) to visit. We construct our attack using this simple knowledge.

Our attack using chosen-plaintext attack. As said above, for any d-dimensional query point q and any encrypted value $E(v)$ where $v = \{\vec{\ell}, \vec{u}\}$, the server returns the values of $sign(u_i - q_i)$ and $sign(\ell_i - q_i)$ (-1 or 1) to the client (for all $i \in [1, d]$). With this knowledge, however, the client can actually recover the value v using a chosen plaintext attack. This means that the client can recover any point p in D, since p's ciphertext $E(p)$ is stored in the leaf-level entries of the shadow index $E(I)$ at the client side. As a result, Hu et al.'s approach cannot conceal the database D from the client (which contradicts their security claims). Our attack works as follows.

Suppose that $v = [\vec{\ell}, \vec{u}]$. Given $E(v)$, the client can perform a binary search along the ith dimension to recover the value ℓ_i (or u_i) in the ith dimension ($i \in [1, d]$). Specifically, if the ith dimension has a domain $[a, b]$, then the client can starts with a random ciphertext $E(x_1)$ for a random $x_1 \in [a, b]$ (he can obtain any such pair $(E(x), x)$ through an oracle from *a chosen-plaintext attack*), and then sends $E(x_1)$ and $E(\ell_i)$ to the server and asks for the value of $sign(x_1, \ell_i)$. If $sign(x_1, \ell_i) = -1$, then the client asks for another ciphertext $E(x_2)$ by choosing a random $x_2 \in (x_1, b]$; if $sign(x_1, \ell_i) = 1$, the client asks for another ciphertext $E(x_2)$ by choosing a random $x_2 \in [a, x_1)$; otherwise, if $E(x_1) = E(\ell_i)$, the client terminates and outputs $\ell_i = x_1$.

Clearly, the client only needs to repeat this procedure for $\log(b - a)$ steps in the worse case to recover the value ℓ_i. Note that this works even for non-discrete domains, as any real valued coordinate still has a fixed level of precision. Hence, with no more than $2d \log n$ steps (where n is the maximum domain size, or bits of precision, for any dimension), the client can fully recover the value v.

This attack shows that in Hu et al.'s method, the client can recover the plaintext for any point in D and any entry in the index I. Using similar techniques, we can also construct an attack where the data owner can recover any query q given only the encrypted version of $E(q)$. In summary, Hu et al.'s method [15] is not secure even if the E used is fully secure.

IV. Hardness of the SNN Problem

Given the observations that none of the existing SNN methods guarantees security in a standard security model, one may wonder if the SNN problem is indeed *hard*. By being hard, we mean that it is at least as hard as some other well-understood cryptography problems that are known having no efficient secure schemes in well-established security models. To answer this question, we will show in this section that the SNN problem is at least as hard as the order-preserving encryption (OPE) problem [5], [6].

An order-preserving encryption $\mathcal{E} : P \to X$ is an encryption that takes a plaintext in a domain P and outputs a ciphertext in a domain X, such that for any $p_1, p_2 \in P$, one can determine if $p_1 > p_2$ or $p_1 < p_2$ given only $c_1 = \mathcal{E}(p_1)$ and $c_2 = \mathcal{E}(p_2)$. (Note that deciding whether $p_1 = p_2$ can be reduced to checking whether both $p_1 > p_2$ and $p_1 < p_2$ are false.) In other words, one can understand an OPE encryption scheme as a set of functions $\{\mathcal{E}, \mathcal{E}^{-1}, \text{op}\}$, such that:

$$\text{op}(c_1, c_2) = 1 \text{ iff } p_1 < p_2, \;\; \text{op}(c_1, c_2) = -1 \text{ iff } p_1 > p_2, \quad (4)$$

where $\text{op}(\cdot)$ is a polynomial operation that does not involve (or have any knowledge of) the decryption function $\mathcal{E}^{-1}$.

The concept of an OPE scheme was first proposed in the database community [1]. The solution, however, did not come with a rigorous analysis of its security guarantee. It was until the efforts from the cryptography community [5], [6] that we understand the hardness of constructing a truly secure OPE scheme. Specifically, Boldyreva et al. [6] have shown that it is impossible to construct a secure OPE in standard security

models (such as indistinguishability against chosen-plaintext attack, a.k.a. IND-CPA). Interested readers are referred to [5], [6] for details. The critical message from [5], [6] is as follows:

Theorem 1: [from [5], [6]] A truly secure OPE does not exist in standard security models, such as IND-CPA. It also does not exist even in much relaxed security models, such as the indistinguishability under ordered chosen-plaintext attack security model (a.k.a. IND-OCPA).

We can establish the hardness of the SNN problem using this negative result. We will show that given $E(q)$, designing an SNN method to find q's (encrypted) exact nearest neighbor from an encrypted database $E(D)$ is as hard as designing a secure OPE in standard security models. For convenience, let $\mathrm{nn}(q, D)$ denote the nearest neighbor of q in D.

The reduction. Suppose that we have a d-dimensional database D that contains N points $p_1, \ldots, p_N$, and an encryption scheme E that is secure under a standard security model M. We use E^{-1} to denote the decryption function of E, and $E(D)$ to denote the set $\{E(p_1), \ldots, E(p_N)\}$. Assume that we can construct a truly secure SNN method that is able to find exactly $E(\mathrm{nn}(q, D))$ efficiently given $E(q)$ and the encrypted database $E(D)$ alone. We denote this polynomial method as $\mathcal{B}$ and formally define it as:

$$\mathcal{B}(E(q), E(D)) \rightarrow E(\mathrm{nn}(q, D)), \mathcal{B} \text{ does not use } E^{-1}. \quad (5)$$

If such a method $\mathcal{B}(\cdot)$ exists, we can construct an OPE $\mathcal{E}(\cdot)$ in the same security model M. Our construction is as follows.

Suppose that the message space for the OPE $\mathcal{E}(\cdot)$ is the same as the encryption scheme E above, which is represented by N values $\{m_1, \ldots, m_N\}$, and without loss of generality, $m_1 < m_2 < \cdots < m_{N-1} < m_N$. Our first step in constructing $\mathcal{E}(\cdot)$ is to map these values to a set of one dimensional points $D = \{p_1, \ldots, p_N, p_{N+1}\}$ using a special random hash function h, such that for any $i \in [1, N+1]$, p_i is a random value subject to the following constraints:

1) $p_i \in \mathbb{Z}^+$ for $i \in [1, N+1]$;
2) $h(m_i) = p_i$ for $i \in [1, N]$;
3) $p_i < p_j$ for any $i, j \in [1, N+1]$ iff $i < j$;
4) $p_{i+1} - p_i < p_i - p_{i-1}$ for any $i \in [2, N]$.

Lemma 1: For any message space $\{m_1, \ldots, m_N\}$, the above hash function $h(\cdot)$ guarantees that:

$$\mathrm{nn}(h(m_i), D) = h(m_{i+1}) \text{ for any } i \in [1, N-1], \text{ and}$$
$$\mathrm{nn}(h(m_N), D) = p_{N+1}, \mathrm{nn}(p_{N+1}, D) = p_N = h(m_N). \quad (6)$$

Lemma 1 indicates that, for any two consecutive values m_i and m_{i+1} in the message space of $E(\cdot)$, the hash value of m_{i+1} is the NN of the hash value of m_i. In addition, for the maximum element m_N in the message space of $E(\cdot)$, the NN of its hash value is the maximum element in the output space of $h(\cdot)$. Figure 1 shows an example of the hash function $h(\cdot)$.

We are now ready to present our OPE. For any m_i from a message space $\{m_1, \ldots, m_N\}$, we define an encryption scheme $\mathcal{E}$:

$$\mathcal{E}(m_i) = E(h(m_i)) = E(p_i), \ \mathcal{E}^{-1}(c) = h^{-1}(E^{-1}(c)), \quad (7)$$

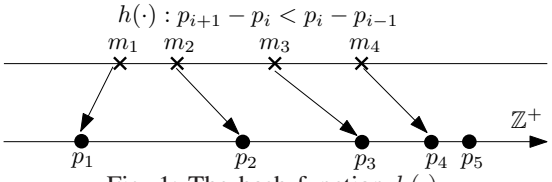


Fig. 1: The hash function $h(\cdot)$.

where $h^{-1}(h(m_i)) = m_i$. The secret in $\mathcal{E}$ contains both the secret of E and the random mapping h, which easily allows us to construct its decryption function $\mathcal{E}^{-1}$ as above. We have the following lemma about $\mathcal{E}$.

Lemma 2: For any $i \in [1, N-1]$ and any ciphertext $\mathcal{E}(m_i)$, we have $\mathcal{B}\big(\mathcal{E}(m_i), E(D)\big) = \mathcal{E}(m_{i+1})$, $\mathcal{B}\big(\mathcal{E}(m_N), E(D)\big) = E(p_{N+1})$, and $\mathcal{B}\big(E(p_{N+1}), E(D)\big) = E(p_N) = \mathcal{E}(m_N)$.

By Lemma 2, if there exists a secure SNN method $\mathcal{B}(\cdot)$ based on the encryption scheme $E(\cdot)$, then we can incorporate $\mathcal{B}(\cdot)$ with $\mathcal{E}(\cdot)$ (i.e., an encryption scheme that combines $E(\cdot)$ and the hash function $h(\cdot)$) to construct a secure approach for searching *successors*. In particular, given the ciphertext $\mathcal{E}(m_i)$ of any non-maximum element m_i in the message space of $\mathcal{E}(\cdot)$, we can invoke $\mathcal{B}(\cdot)$ to compute the ciphertext of m_{i+1}, where m_{i+1} is the smallest element in the message space that is larger than m_i.

Algorithm 1: op($c = \mathcal{E}(m_i)$, $z = \mathcal{E}(m_j)$)

```
i = traverse(c);    /* see Algorithm 2 */
j = traverse(z);    /* see Algorithm 2 */
if i < j then return 1; else return −1;
```

Algorithm 2: traverse(c)

```
set γ = 0;
let t = B(c, E(D)); t' = c
while B(t, E(D)) ≠ t' do
    let t' = t and t = B(t, E(D));
    γ = γ + 1;
return N − γ;    /* B(·), E(D) are global inputs */
```

Next, we will show that $\mathcal{E}(\cdot)$ is an OPE, by proving that there exists an operator $op(\cdot)$ that satisfies the Equation 4. Algorithm 1 presents the details of our formulation of $op(\cdot)$. Note that the algorithm only uses $E(D)$ and $\mathcal{B}$ without any knowledge about the secret of $\mathcal{E}$ (which includes both the secret of E and the random mapping h). Algorithm 1 also uses Algorithm 2 as its building block. Given a ciphertext $c = \mathcal{E}(m_i)$, Algorithm 2 outputs the index value i for the plaintext m_i. Hence, op$(\cdot)$ in Algorithm 1 simply compares the two index values of the two input ciphertexts.

Specifically, the idea in Algorithm 2 is as follows. By Lemma 2, $\mathcal{B}(c, E(D)) = \mathcal{B}(\mathcal{E}(m_i), E(D))$ outputs $\mathcal{E}(m_{i+1})$ (for $i \in [1, N-1]$). Now, we can repeatedly apply $\mathcal{B}(\cdot)$ γ times on its result until we hit $\mathcal{E}(m_N) = E(p_N)$; clearly, $i = N - \gamma$. The only challenge left is: how can we check if we have reached $\mathcal{E}(m_N) = E(p_N)$? To explain how we address this, observe that, by Lemma 2, we have $\mathcal{B}\big(\mathcal{E}(m_N), E(D)\big) =$

$E(p_{N+1})$ and $\mathcal{B}\big(E(p_{N+1}), E(D)\big) = E(p_N) = \mathcal{E}(m_N)$, i.e., once we hit $\mathcal{E}(m_N)$ for the first time, subsequent calls to $\mathcal{B}(\cdot)$ on its result will bounce between $E(p_{N+1})$ and $\mathcal{E}(m_N)$. Furthermore, among all possible ciphertexts $\mathcal{E}(m_1) = E(p_1), \ldots, \mathcal{E}(m_N) = E(p_N), E(p_{N+1})$, this mini-loop only happens between $\mathcal{E}(m_N)$ and $E(p_{N+1})$ (when we repeatedly apply $\mathcal{B}(\cdot)$ to its own output).

Thus, if we record the current value of t as t' before we set $t = \mathcal{B}(t, E(D))$ in line 4 in Algorithm 2 (same idea in line 2), then when t becomes $\mathcal{E}(m_N)$, line 4 would first set $t' = t = \mathcal{E}(m_N)$, and then set $t = \mathcal{B}(t, E(D)) = \mathcal{B}(\mathcal{E}(m_N), E(D)) = E(p_{N+1})$. After that, $\mathcal{B}(t = E(p_{N+1}), E(D))$ outputs $E(m_N) = t'$ again, in which case t and t' would satisfy $\mathcal{B}(t, E(D)) = t'$; this is precisely the condition that terminates the loop in line 3.

Finally, for $c = \mathcal{E}(m_i)$, it is easy to verify that this termination condition is first met when we have iterated t through $\mathcal{E}(m_{i+1})$, ..., and $\mathcal{E}(m_N)$ exactly once (by the mini-loop observation above). Thus, the running time of Algorithm 2 is at most $O(NZ)$ where Z is the running time of the SNN method $\mathcal{B}(\cdot)$. This indicates that the cost of the Algorithm 1 for op is also $O(NZ)$.

Theorem 2: Let $\{m_1, \ldots, m_N\}$ be any message space and $m_i < m_j$ if $i < j$. Let $\mathcal{E}(m_i) = E(h(m_i))$ and $\mathcal{E}^{-1}$ be as defined in (7), using the hash function h and the secure encryption scheme E from the SNN method $\mathcal{B}(\cdot)$. Define $\mathrm{op}(\cdot)$ as shown in Algorithm 1. Then $(\mathcal{E}, \mathcal{E}^{-1}, \mathrm{op})$ is an OPE scheme that is secure in any security model M in which E is secure.

Finally, by Theorems 1 and 2, our conclusion is:

Theorem 3: It is impossible to construct a secure SNN method $\mathcal{B}(\cdot)$ satisfying (5) in standard security models, such as IND-CPA. It is not even possible to construct such an $\mathcal{B}(\cdot)$ in much relaxed security models such as IND-OCPA.

V. Partition Based SNN Method

As shown in Theorem 3, given only an encrypted NN query $E(q)$ and an encrypted database $E(D)$, there is no SNN method that can pinpoint the NN of q in D. To circumvent this impossibility result, a natural idea is to devise an SNN method that does *not* exactly retrieve the NN of q. For example, the server may answer an SNN query by returning the encrypted database $E(D)$ as a whole to the client, after which the client can decrypt $E(D)$ and compute the answer to the query locally. This naive approach is clearly secure (as secure as E), but it is highly inefficient as it requires transferring $E(D)$ to the client. To improve, we propose to partition D into small groups and store the encrypted version of each group on the server, such that any SNN query can be answered by returning *one* encrypted group instead of the whole encrypted database. Specifically, our partitioning of D is based on the *Voronoi diagram* of D, as explained in the following.

A. Secure Voronoi Diagram

Given a multi-dimensional point database D with $|D| = N$, a *Voronoi diagram* of D is a decomposition of the space Ω in which the points in D are defined. The diagram consists of n disjoint *Voronoi cells*, each of which is associated with a point in D (referred to as the *owner* of the cell). If a point p is the owner of a cell c, then c equals the union of all points in Ω that are closer to p than to any other points in D. Thus, if a query point q falls in c, then its nearest neighbor in D is p. For example, Figure 2 illustrates the Voronoi diagram of a database with 16 two-dimensional points.

The Voronoi diagram of points in D can induce a partition of D for SNN. Specifically, we can impose a square grid on the Voronoi diagram, and then construct an overlapping partition of D, such that each element of the partition (i) corresponds to a grid cell B and (ii) consists of the owners of all Voronoi cells that overlap with B. For example, in Figure 2, the partition element (i) G_1 corresponds to the grid cell B_1, and (ii) G_1 consists of ten points (i.e., p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8, and p_{10}), since B_1 overlaps with the Voronoi cells owned by those ten points. Observe that, if a query point falls in a grid cell B_i, then its nearest neighbor must be a point in the partition element G_i associated with B_i.

Given the aforementioned partition of D, the data owner pads each partition element to the same size, and then encrypts them separately (with the same key) and gives each element a random identifier. (The padding procedure ensures that the encrypted partition elements cannot be distinguished by their sizes). After that, the data owner sends all encrypted partition elements and their associated identifiers to the server, and informs the client about the description of the square grid and the identifier of each partition element. Whenever the client has an SNN query q, she first identifies the grid cell B that contains q, and then retrieves the identifier i of the partition element that corresponds to B. Then, the client asks the server to return the encrypted partition element whose identifier equals i (notice that this partition element must contain the nearest neighbor of q). Upon receiving the partition element, the client decrypts it and computes the answer to q locally. Intuitively, this SNN method is secure, as it allows the server to learn nothing but the identifier of the returned encrypted partition, which is randomly generated. We provide the formal security proof in Section V-C.

The above partition scheme, albeit simple and secure, incurs significant space overhead (for the server) and communication cost (for the client). To understand this, recall that each partition element needs to be padded to the same size before encryption. Let s_{max} be the number of points in the largest partition element. Then, after padding, the size of each partition element is s_{max}. Assume that the size of the encryption of a message is (roughly) linearly dependent (or in a stair-case fashion) on the size of the message, which is the case for most encryption functions, the server requires $O(k \cdot s_{max})$ space to store all encrypted partition elements (where k denotes the total number of partition elements), and the client pays $O(s_{max})$ communication cost for each SNN query. Ideally, we would prefer a partition scheme that ensures $s_{max} = N/k$, in which case both the server's space overhead and the client's communication cost are minimized. This, however, would

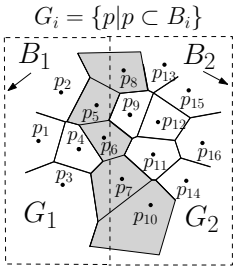
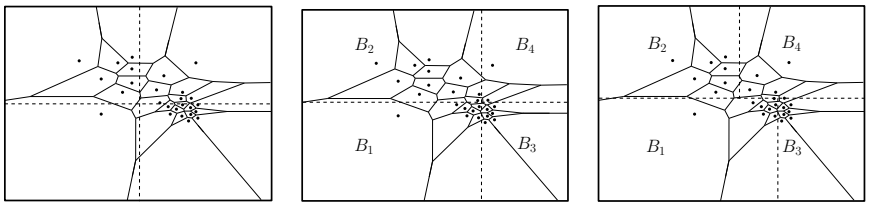


Fig. 2: The SVD method.



(a) Square Grid: SG. (b) MinSG. (c) MinMax.

Fig. 3: Different partitioning schemes, $k = 4$.

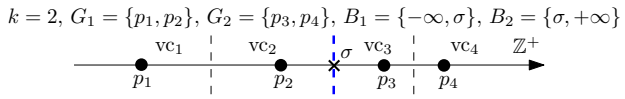


Fig. 4: Partition Scheme for One-dimensional Data.

require that each partition element contains equal number of points, which is rather unlikely under the square grid partition scheme. Specifically, as real datasets are often skewed, some cell in a square grid may intersect a significantly larger number of Voronoi cells than other grid cells. This leads to a large s_{max}, which results in significant space and communication overheads. To remedy this deficiency, we propose improved schemes that adaptively partition the data space to minimize s_{max}, as will be discussed in Section V-B.

Remark: The partition scheme of imposing a square grid over the Voronoi diagram was also used by Ghinita et al. [10] in the context of *private information retrieval (PIR)*. We will discuss the difference between SNN and PIR in Section VII. We dub this scheme the *SG* (Square Grid) partition.

B. Improved Partition Schemes

In what follows, we present partition schemes that aim to minimize s_{max}, i.e., the number of points in the largest partition element. We focus on the case when D is one- or two-dimensional, as (i) SNN queries are particularly important for one- or two-dimensional spatial data, and (ii) constructing Voronoi diagrams is computationally challenging on data with dimensionality over 2 (as each Voronoi cell would become a complex polytope).

1) One Dimension: In one dimension, there is an optimal partition scheme that generates disjoint and equal-size partition elements. To explain this, observe that the Voronoi cells of any one-dimensional dataset are one-dimensional intervals, such that (i) any two intervals are disjoint, and (ii) the union of all intervals equals the data space. For example, Figure 4 shows the Voronoi cells $(vc_1, \ldots, vc_4)$ of a one-dimensional dataset that contains four points p_1, p_2, p_3, p_4.

Assume that we aim to generate a partition of D with k elements. Then, it suffices to divide the data space into k disjoint intervals, such that each interval contains N/k Voronoi cells of D (where N is the number points in D). The owners of the Voronoi cells in each interval can be taken as partition element, which leads to an optimal partition scheme where $s_{max} = N/k$. For instance, if we are to generate a partition with two elements on the dataset in Figure 4, then we can put p_1, p_2 into a partition element, and p_3, p_4 into the other.

2) Two Dimensional Data: The case for two-dimensional data is much more complicated than the one-dimensional case, which renders it difficult to compute an optimal solution efficiently. This motivates us to derive two partition schemes based on heuristics, as will be explained in the following.

The MinSG scheme. This partition scheme adopts a greedy approach to generate a grid where the sizes of cells vary in a manner that adapts to the distribution of points in D. We start with a grid that contains only one cell Ω that cover the entire data space. Then, we iteratively cut the grid cells into smaller ones using either horizontal (or vertical) lines through $[-\infty, +\infty]$ along the x-axis (or the y-axis). The process is conducted until the desired number of grid cells is met. After that, we generate the partition by identifying the owners of the Voronoi cells that intersect with each grid cell (as with the case of a square grid in the SG method).

The effectiveness of the above algorithm relies on how we choose the horizontal (or vertical) line to cut the grid cells. As we aim to minimize s_{max}, we propose to choose a line cutting through the grid cell B_{max} that determines s_{max}, i.e., the grid cell that intersects the largest number of Voronoi cells. The intuition is that, when B_{max} is split into two smaller cells B_α and B_β, each of the two cells may intersect a smaller number of Voronoi cells, which leads to a decrease of s_{max}. To maximize the decrease of s_{max}, we should minimize $\max\{|s_\alpha|, |s_\beta|\}$, where $|s_\alpha|$ ($|s_\beta|$) denotes the number of Voronoi cells that intersects B_α (B_β). Choosing a line that minimizes $\max\{|s_\alpha|, |s_\beta|\}$, however, is challenging given that there exists an infinite number of lines that cut through B_{max}. To remedy this problem, we have a key observation as follows.

Lemma 3: $\max\{|s_\alpha|, |s_\beta|\}$ is minimized only if the cutting line passes through (i) a vertex of a Voronoi cell or (ii) the intersection point between the boundary of B_{max} and a face of a Voronoi cell.

Let $[x_l, x_u]$ and $[y_l, y_u]$ be the projections of B_{max} onto the x- and y-axes, respectively. Let V_1 be a set that contains any vertex of any Voronoi cell with its x-coordinate in $[x_l, x_u]$ or its y-coordinate in $[y_l, y_u]$. Let V_2 be a set that contains any intersection point between the boundary of B_{max} and a face of any Voronoi cell. By Lemma 3, we can identify the cutting line that minimizes $\max\{|s_\alpha|, |s_\beta|\}$, by (i) inspecting all horizontal or vertical lines that pass through the points in $V_1 \cup V_2$ and (ii) choosing the line that leads to the smallest $\max\{|s_\alpha|, |s_\beta|\}$.

As mentioned, the above splitting procedure is applied

iteratively based on the grid cell that intersects the largest number of Voronoi cells, until the number of grid cells reaches a pre-defined threshold k. Since each cutting line that we use would span the whole x- or y-axis, it may split more than one grid cells, and hence, when the algorithm terminates, the total number of grid cells may be more than k. Whenever this happens, we would iteratively merge some neighboring cells without affecting the size of the largest partition, until only k cells are left. We omit such technical details for brevity.

We refer to the aforementioned partition scheme as the MINSG (*Min*imum *S*pace *G*rid) method. For example, Figure 3(b) shows the results of applying MINSG over the same dataset in Figure 3(a).

It can be verified that the number of iterations performed by MINSG is no more than $k-1$ (since each iteration increases the total number of grid cells by at least one). Furthermore, the number of cutting lines that MINSG needs to inspect in each iteration is $O(N)$, since (i) the total number of vertices in a Voronoi diagram is $2N-5$ (for $N \geq 3$), i.e., $|V_1| = 2N-5$, (see Theorem 7.3, [8]); and (ii) each Voronoi cell, being a convex polygon, can intersect the boundary of a grid cell at no more than 8 points, i.e., $|V_2| \leq 8N$. For each candidate cutting line ℓ, we need to identify the Voronoi cells intersecting ℓ and to incrementally update the number of Voronoi cells that overlap with each grid cell, which incurs $O(N)$ cost in the worst case. Therefore, the time complexity of MINSG is $O(kN^2)$. In practice, however, the running time of MINSG is often not quadratic to N, since the sizes of V_1 and V_2 are often much smaller than N.

The MinMax scheme. The MINSG method tries to minimize the maximum element size in the partition, by iteratively splitting the largest partition element. Nevertheless, the split is induced by a horizontal or vertical line that spans the whole x- or y-axis, which often incurs unnecessary split of other partition elements. For example, consider the grid partition in Figure 3(b), where the grid cell B_1 intersects the largest number of Voronoi cells. If we are to split B_1 with an vertical line ℓ, then the cell B_2 would be split into two smaller cells by ℓ as well, even though the split of B_2 does not decrease the maximum element size in the induced partition.

To address this problem, we propose to improve MINSG by splitting, in each iteration, nothing but the grid cell that induces that largest partition element. Specifically, in an iteration where the large partition element is induced by grid cell B_{max}, we would split B_{max} using a horizontal (vertical) line *segment* ℓ' whose projection on the x-axis (y-axis) equals B_{max}'s projection on the same axis. Furthermore, the line segment is selected among those that pass through (i) a vertex of a Voronoi cell or (ii) the intersection point between a face of a Voronoi cell and the boundary of B_{max} – the correctness of this approach can be shown by a result similar to Lemma 3. The rest of the algorithm is the same as in MINSG. We refer to this improved method as the MINMAX (*Min*imum *Max* partition) method. Figure 3(c) shows the results of applying MINMAX on the same dataset in Figure 3(a).

Observe that each iteration of MINMAX increases the number of grid cells by one. Thus, the total number of iterations performed by MINMAX is $(k-1)$. Each iteration inspects $O(N)$ cutting line segments, and examines $O(N)$ Voronoi cells for each line segment. Therefore, the time complexity of MINMAX is $O(kN^2)$, as with the case of MINSG. But, for similar reasons, its running time in practice is much better.

Remark. No matter which partition scheme is used, the data owner does not send the k partition elements to the client, which have a total size of N points. Instead, he only needs to send the description of the grid constructed, and the associated (randomly generated) identifier for each grid cell, which only has a size of $O(k)$ for all three methods, SG, MINSG, and MINMAX.

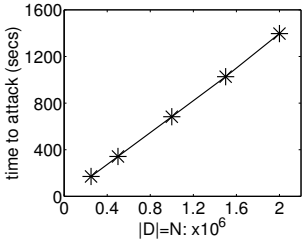
C. Security Analysis

No matter which partition scheme is used, the SVD method only releases the encryptions of all partition elements to the server, using an encryption scheme E that is proven secure in a standard security model M, and their associated (randomly generated) identifiers. Furthermore, during query processing, the client sends only the identifier for the partition element whose corresponding grid cell contains the query point q, to the server. Thus, the server learns nothing but an id number (randomly generated by the data owner for each partition element) in answering a query. Formally:

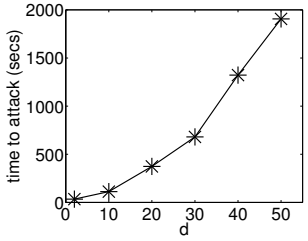
Theorem 4: If E is a secure encryption scheme in a standard security model M, e.g., indistinguishability under chosen plaintext attack (IND-CPA), then the SVD method is as secure as E in the same model M with respect to a single query.

Further improvement of security. The above analysis considers a powerful adversary that is able to obtain a large number of plaintext-ciphertext pairs (in either chosen plaintext attack model or chosen ciphertext attack model), but implicitly assumes that the adversary has no knowledge of how the user's queries are distributed. When this assumption does not hold, the adversary may infer information by exploiting the correlation between different encrypted queries from the user. For example, assume that the adversary knows in advance that most of the queries from the user would have a query point in a region R. Then, the adversary can keep track of the encrypted partition element retrieved by each user query over a long period, after which the adversary can link R to the partition element that is retrieved most frequently.

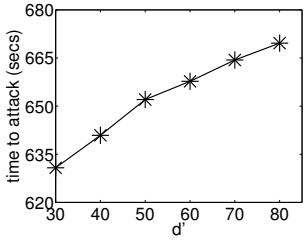
To guard against such attacks, we can extend our solution by adopting *private information retrieval (PIR)* techniques [10]. In particular, for any SNN query q, the user can first identify the encrypted partition element that contains the answer to q; and then, the user can invoke a PIR protocol to retrieve the encrypted partition element from the server, without allowing the server to learn which partition element is returned. But notice that existing PIR techniques either incur considerable overhead in query processing [10] or require specialized secure hardware that is expensive [17], i.e., the improved security guarantee would come at a cost.



(a) Vary $|D|$.



(b) Vary d.



(c) Vary d', $d = 30$.

Fig. 5: The attack on solution by Wong et al. [21]

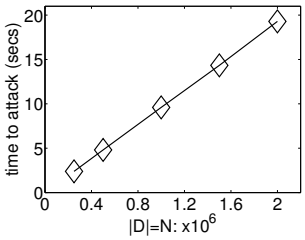


Fig. 6: The attack on solution by Hu et al. [15], varying $|D|$.

Return a partition element vs. return $\mathrm{nn}(q, D)$**.** One may wonder if the SVN method "leaks" too much information to the client about the database D, by returning a partition element (i.e., a number of points) for each query. We argue that this should not be a concern at all: if one allows the client to learn the nearest neighbor of a query point of her choice (by any SNN methods), the client can efficiently reconstruct the entire database D anyway, i.e., there is no way of hiding the database D from the client if one wants to support client's nearest neighbor queries. We show why this is the case in Appendix B in our online technical report [22].

VI. Experiment

Our implementations were achieved in C++. We used the Qhull library to find the Voronoi diagram for a dataset D, and the latest Crypto++ library for any standard encryption schemes. All experiments were performed on a Linux machine with an Intel Xeon 3.07GHz CPU and 12GB memory.

Datasets. When investigating our attacks to existing schemes in different dimensions, we generate random points following either a *random cluster* or a *uniform* distribution. Based on the construction of our attacks, the distribution of the points does not affect the efficiency of our attacks.

For SVD methods, we focus on 2-dimensions with two real datasets, which are points of interest in California (CA) and Texas (TX) from the OpenStreetMap project. In each dataset, we randomly select 2 million points to create the largest dataset $D_{\max}$ and form smaller datasets based on $D_{\max}$. We also make sure that smaller datasets are always subsets of larger datasets, in order to isolate the impact of $|D|$.

Setup. In the investigation of our attacks, since the distribution of the data does not introduce a noticeable difference, our default distribution is *uniform*. In the scheme from Wong et al. [21], we set the default values of d and d' as $d = 30$ and $d' = 60$. In the scheme from Hu et al. [15], we only report the results from $d = 2$ for brevity.

In the study of the SVD method, the default values are: $|D| = 10^6$ and $k = 625$ (number of partition elements). The default dataset is CA. By default, we used the AES encryption scheme with a key size of 256 bits and a block size of 256 bits. The trends from other secure encryption schemes are similar. Any secure public-key or symmetric-key encryption scheme can be used to construct a SVD method.

In all experiments, unless otherwise specified, when we vary the value of one parameter in concern, *we keep all other parameters at their default values.*

A. Attacks to existing methods

We first studied the efficiency of our attack on the SNN method by Wong et al. [21], and the results are given by Figure 5. As shown in Section III-A, our attack only needs d query points and their encryption to form a linear system of d linear equations with d unknowns to solve one data point $p \in D$.

Thus, the cost of our attack should be linear to both $|D|$ and d, which is reflected in Figures 5(a) and 5(b). Our attack is extremely efficient. For example, Figure 5(a) shows that it takes less than 23 minutes to recover a database with 2 million points in 30 dimensions and also with 30 artificial dimensions. Lastly, the artificial dimensions d' introduced in the solution of Wong et al. [21] is not significant to our attack, as seen in Figure 5(c). When d' goes from 30 up to 80, the overhead introduced to the overall time to attack is less than 40 seconds when the total time to attack is 600 some seconds.

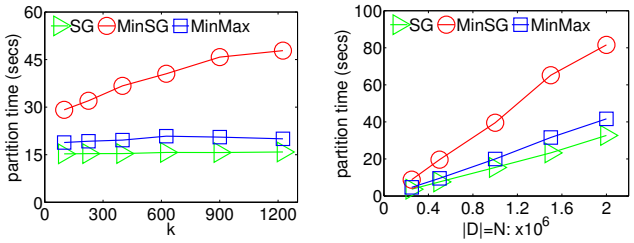
Next, focusing on $d = 2$ and a domain size of 10^9 in both dimensions, Figure 6 shows that our attack to the solution by Hu et al. [15] is extremely efficient when we vary $|D|$. For a database with 2 million points, each with 1 billion possible values in either dimension, our attack takes less than 20 seconds to recover all 2 million points! Our attack is linear to both d and $|D|$, since its cost to recover one point in D is only $O(d \log n)$ where n is the maximum domain size in any one dimension according to our analysis in Section III-B.

Finally, both attacks can be easily made parallel, as they recover each point in D independently.

B. Evaluation of the SVD method

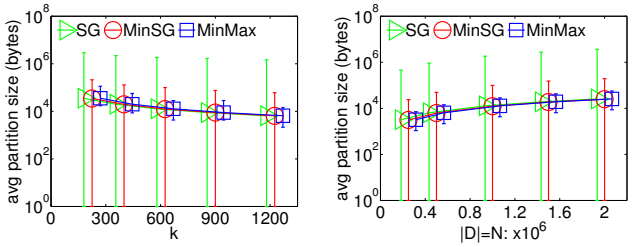
We next evaluate the efficiency of the new SVD method, when we instantiate it with three different partitioning schemes introduced in Section V. To differentiate them, we simply refer to the resulting SVD methods by the names of the partitioning methods used.

Preprocessing cost. For the data owner, there are two major steps: partition and encryption. Both are mainly affected by the number of grid cells k (which is also the number of partition elements) and the size of the database. Figure 7(a) shows that the partition costs in both MinSG and MinMax increase linearly with k, however, the cost of MinSG increases much faster than that in MinMax, which increases very slowly (almost unnoticeable). This is due to the optimization in MinMax (compared to the cutting lines in MinSG) to limit any cutting line to be strictly within the cell to split.



(a) vary k. (b) vary $|D|$.

Fig. 7: Running time of the partition phase.



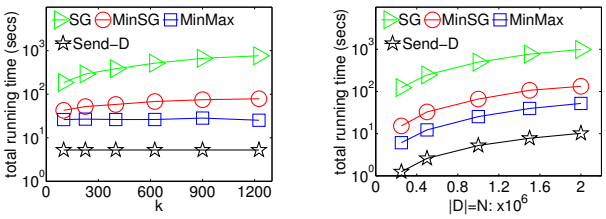
(a) vary k. (b) vary $|D|$.

Fig. 8: Partition size in different methods.

The partition cost of SG is constant regardless k, since it calculates the unit length of each cell constantly once k is given. MINMAX is extremely efficient; its partition cost is almost as good as the partition cost of SG. It produces 1,225 partitions for 1 million points in just 20 seconds.

Next, Figure 7(b) shows the partition cost when we vary the size of the database from 0.2 million to 2 million. The partition cost in SG clearly should be linear to $N = |D|$ once it has figured out the unit length of each cell, which is observed in Figure 7(b). Interestingly enough, even though our analysis in Section V-B indicates that the worst case complexity for both MINSG and MINMAX is $O(kN^2)$. In practice, both their costs are in fact only linear to N. This is because that the bad case in our analysis, where a cutting line intersects with all $O(N)$ Voronoi cells, is almost impossible in practice. Instead, for both MINSG and MINMAX, in any step a cutting line typically only intersects with a constant number of Voronoi cells. This means that both their costs should be only $O(kN)$ (for a maximum of k possible steps). Also, clearly a cutting line in MINSG expects to intersect with more Voronoi cells than that in MINMAX in any step. Hence, we also see a higher cost in MINSG than the cost of MINMAX, and a faster pace of increase in cost when $|D|$ increases. Figure 7(b) indicates that MINMAX is almost as efficient and scalable as SG. When $|D|$ changes from 0.2 to 2 million points, the partition cost of MINMAX only increases from 5 seconds to 40 seconds.

We then examine the sizes of the partition elements from different methods *before applying the random padding operation*. Due to the random padding operation to "inflate" every partition element with random bytes to the size $s_{\max}$ of the maximum partition element, two values are critical: $s_{\max}$, which decides the storage cost at the server and the communication cost of every query, and the variance of the sizes of partition elements, which decides the overhead of the total "inflation". Figure 8 plots the average size of partition elements along with $s_{\max}$ and $s_{\min}$ (the size of the minimum partition element), as the error bar, w.r.t. k and $|D|$.



(a) vary k. (b) vary $|D|$.

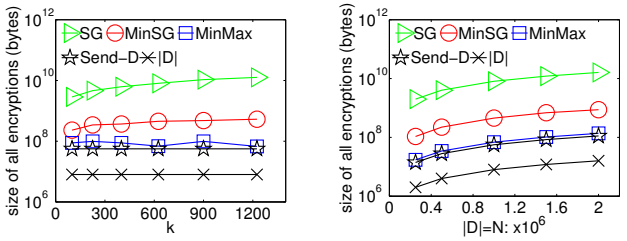
Fig. 9: Total running time of the preprocessing step.

All methods produce partitions that have a similar average size. However, SG always has the largest values in both $s_{\max}$ and the variance, due to its complete ignorance of the data distribution. MINSG does significantly reduce the value of $s_{\max}$, which is expected since its design is to greedily split the partition element with the maximum size. However, following our analysis in Section V-B, due to the long extent of its cutting lines (always from $[-\infty, +\infty]$), its splitting method will potentially lead to partitions of very small size, as observed in Figure 8. The design of MINMAX makes further improvement to MINSG, so it further reduces $s_{\max}$ considerably. It also eliminates very small partition elements by limiting a cutting line to be only within the extent for the cell to split (the cell for the current maximum partition element). Hence, MINMAX always produces a balanced partition. As a result, its $s_{\max}$ is very close to the average size of all partition elements and the variance is also small, as seen in Figure 8.

Finally, not surprisingly, for all partition methods, the values for both the average size and the $s_{\max}$ reduce when more grid cells are used (Figure 8(a)), and increase when D grows (Figure 8(b)). Nevertheless, MINMAX always produces highly balanced partitions with the smallest (much smaller than that from the other two methods) $s_{\max}$ values.

Figure 9 reports the *total running time for the preprocessing step*, which includes the costs of both the partition and the encryption (as well as the Voronoi diagram construction and the random padding operation, whose costs are small compared to partition and encryption costs). For reference, we also include the preprocessing cost from the naive method *Send-D* (see the first paragraph in Section V), whose preprocessing cost is to encrypt D in its entirety as one message.

What is interesting to observe is that, even though SG is fastest in producing its partitions, it becomes the slowest method overall. This is explained by the fact that partitions produced by SG suffer from the largest variance and the largest $s_{\max}$ value. After the random padding operation, all partitions share the same size as the maximum partition element. Hence, the value of $s_{\max}$ decides the total encryption cost when there are equal number of partition; and the variance in partition size decides the overhead introduced to the encryption step by the "inflation" of random bytes. Given the results in Figure 8, it is not surprising to see that SG becomes the slowest method, and MINMAX becomes the fastest method (especially when MINMAX also enjoys a partition cost that is almost as good as the partition cost in SG, see Figure 7).



(a) vary k. (b) vary $|D|$.

Fig. 10: Total size of all encrypted partition elements.

Thus, it is natural to see a linear increase w.r.t. k and $|D|$ in the total running time in producing all encrypted partition elements for all SVD methods. However, as shown in Figure 9, MINMAX is at least one order of magnitude faster than SG, and about 3-6 times faster than MINSG. In fact, MINMAX only takes less than 90 seconds to produce all encrypted partition elements with $k = 625$ over a database with 2 million points as shown in Figure 9(b). Of course, *Send-D* is fastest, but MINMAX's overhead compared to *Send-D* is small.

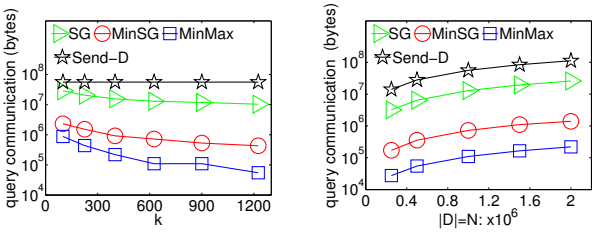
We also examined the total size of all encrypted partition elements. It affects both the communication cost from the data owner to the server and the storage cost at the server. After the random padding operation, each partition element has a size of $s_{\max}$; hence, this value is simply $ke_{\max}$, assuming $e_{\max}$ is the size of the ciphertext for a message of size $s_{\max}$ encrypted under the secure encryption scheme E used.

Figure 10 shows the total size of all encrypted partition elements for all methods. We can view *Send-D* as a method with only one partition element to encrypt, which is D itself. For reference, we also include the size of D. Clearly, the total size only increases linearly with $|D|$ in all methods; it also increases linearly with k in all SVD methods. Among the three SVD methods, MINMAX is the clear winner, which again is due to the smallest $s_{\max}$ value it achieves (see Figure 8). All methods introduce a owner-server communication and server storage overheads compared to using the database D itself. However, MINMAX's overhead is almost as small as that from the naive method *Send-D* which has the smallest such overheads one can hope for.

Finally, the client's storage cost is dependent *only on the size of the description for all grid cells* as shown in Section V-C, which is $O(k)$ for all three SVD methods, regardless the size of the database and the sizes of partition elements! Since typical k values are very small compared to the database size (having few hundred grid cells is good enough for a few million points), this cost is almost negligible.

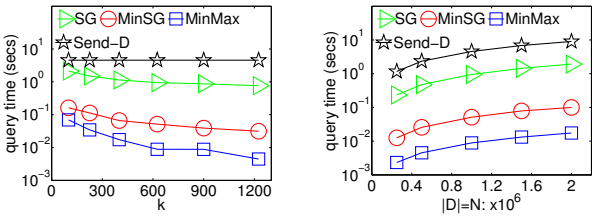
Lastly, we do not observe any significant difference from the TX datasets. Hence, we omit all results from TX for brevity.

Query processing cost. Recall that for a query q, the client locally figures out the identifier for the grid cell that contains q, using the description of all grid cells she has; then she simply requests the server to return the corresponding, encrypted partition element. But due the random padding operation, each encrypted partition element has the same size, $e_{\max}$, which is the size of the encryption for a message of size $s_{\max}$ (the size of the maximum partition element) under the secure encryption scheme E used to construct the SVD method.



(a) vary k. (b) vary $|D|$.

Fig. 11: Query communication cost.



(a) vary k. (b) vary $|D|$.

Fig. 12: Query time for different methods.

That said, the server-client communication cost is always $e_{\max}$ for any query in a SVD method. In contrast, this cost for *Send-D* is the size of the encryption for the entire database. As a result, the server-client communication cost from all SVD methods are much better than that in *Send-D*, especially for the best SVD method MINMAX, as shown in Figure 11. In fact, MINMAX's query communication cost is 2-3 orders of magnitude smaller than the cost of *Send-D* in most cases!

Next, we examine the query processing cost at the client side. For a SVD method, this consists of two parts: 1) figure out the identifier for the grid cell that contains a query point q, which can be done efficiently using a multi-dimensional index over the description of all grid cells; 2) decrypt the encrypted partition element returned from the server, and find the nearest neighbor of q among points in this partition element. For the native method *send-D*, its query cost is to decrypt the encrypted database, and then find NN of q from D.

For each experiment, we generate 100 random queries and report the average in Figure 12. Figure 12(a) shows that the query costs of all SVD methods reduce when k increases, due to smaller number of points in the partition element returned. However, MINMAX enjoys the best improvement when k increases. This is because we report the average query cost and it produces the most balanced partition elements. All SVD methods outperform the naive method *Send-D* in all k values. In fact, MINMAX is faster by 2-3 orders of magnitude than *Send-D*, and by 1-2 orders of magnitude than SG. When $|D|$ increases, the query time of all methods increases linearly with $|D|$ as shown in Figure 12(b). Nevertheless, MINMAX still has the best performance, which is 3 orders of magnitude faster than *Send-D*, and 2 orders of magnitude faster than SG. When D has 2 million points, with 625 partitions MINMAX's query time at client is as little as 10^{-2} second.

VII. RELATED WORK

Our study falls into the general category of *secure query processing* as discussed in Section I. Existing work have

examined the problems of answering basic SQL queries [11], executing aggregate queries [13], [16], and performing range queries [14], [19], over an encrypted database. As shown by existing studies [14], [15], [19], [21], special treatments are often required for more complex query types, to meet the security requirements and/or achieve better efficiency. In particular, research effort [15], [21] has been made to address the SNN problem; the solutions thus proposed, however, are insecure and can be attacked efficiently, as we have demonstrated.

Secure query processing (including the SNN problem) is related to but different from another well known problem called *private information retrieval (PIR)* [3], [10], [20]. In PIR, the objective is to prevent the server from knowing anything about which records have been retrieved/accessed by a user query, while the server is allowed to know all of the tuples in the database. In particular, using PIR for answering NN queries has been studied in [10], [17], but the solutions thus proposed only protect the privacy of user queries, without preventing the server from knowing the content of the database. For similar reasons, our problem is also different from privacy issues in location-based services [2], [4], [7].

Our reduction from OPE to SNN (which shows the hardness of the SNN problem) leverages on the concept of order-preserving-encryption, which was first proposed in [1]. However, the OPE schemes proposed in [1] did not come with any formal analysis on their security guarantees. It was until recently that the formal security analysis on the general concept of OPE schemes has surfaced from the cryptography community [5], [6]. In particular, it is proven that it is impossible to construct secure OPE schemes in standard security models (see Section IV and [5], [6] for details).

Our partitioning schemes share some similar motivation from the design of indexing methods based on Voronoi diagram, e.g., [18]. However, our methods must adhere to the specific partitioning policy derived based on the particular security and efficiency constraints to the SNN problem, which makes our study here different from existing work. Lastly, as discussed in Section V-A, the basic method SG was used in a different context for answering spatial queries using PIR techniques. However, as shown by our study, we have proposed new partitioning schemes, MINSG and MINMAX that significantly outperform SG in terms of both the size of the maximum partition element and how balance the sizes of different partition elements are.

VIII. CONCLUSION

This work revisits the secure nearest neighbor problem. We show the insecurity of existing solutions, and the hardness of the SNN problem. We then design a new partition-based secure Voronoi diagram (SVD) method. The SVD method is as secure as the encryption function it uses, and any standard secure encryption schemes can be employed by the SVD method. Extensive experiments clearly demonstrate the efficiency and the scalability of the SVD method. Future work include extending our investigation to higher dimensions, k nearest neighbors, nearest neighbors in non-euclidean space (e.g., on road networks), and other similarity search queries.

ACKNOWLEDGMENT

Bin Yao was supported by the NSFC (No. 61202025) and the 863 Program of China (No. 2011AA01A202). Feifei Li was supported by NSF Grant CNS-0831278. Xiaokui Xiao was supported by the Nanyang Technological University's SUG Grant M58020016, and by Agency for Science, Technology, and Research (Singapore) under SERC Grant 102-158-0074.

REFERENCES

[1] R. Agrawal, J. Kiernan, R. Srikant, and Y. Xu. Order preserving encryption for numeric data. In *SIGMOD*, 2004.
[2] C. A. Ardagna, M. Cremonini, E. Damiani, S. D. C. di Vimercati, and P. Samarati. Location privacy protection through obfuscation-based techniques. In *DBSec*, 2007.
[3] D. Asonov and J. C. Freytag. Almost optimal private information retrieval. In *Privacy Enhancing Technologies*, 2002.
[4] C. Bettini, S. Jajodia, P. Samarati, and X. S. Wang, editors. *Privacy in Location-Based Applications, Research Issues and Emerging Trends*, Lecture Notes in Computer Science. Springer, 2009.
[5] A. Boldyreva, N. Chenette, Y. Lee, and A. O'Neill. Order-preserving symmetric encryption. In *EUROCRYPT*, 2009.
[6] A. Boldyreva, N. Chenette, and A. O'Neill. Order-preserving encryption revisited: Improved security analysis and alternative solutions. In *CRYPTO*, 2011.
[7] C.-Y. Chow, M. F. Mokbel, and W. G. Aref. Casper*: Query processing for location services without compromising privacy. *ACM TODS*, 34(4), 2009.
[8] M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars. *Computational Geometry: Algorithms and Applications*. Springer, 3rd edition, 2008.
[9] J. Domingo-Ferrer. A provably secure additive and multiplicative privacy homomorphism. In *ISC*, 2002.
[10] G. Ghinita, P. Kalnis, A. Khoshgozaran, C. Shahabi, and K.-L. Tan. Private queries in location based services: anonymizers are not necessary. In *SIGMOD*, 2008.
[11] H. Hacigümüs, B. R. Iyer, C. Li, and S. Mehrotra. Executing SQL over encrypted data in the database service provider model. In *SIGMOD*, 2002.
[12] H. Hacigümüs, B. R. Iyer, and S. Mehrotra. Providing database as a service. In *ICDE*, 2002.
[13] H. Hacigümüs, B. R. Iyer, and S. Mehrotra. Efficient execution of aggregation queries over encrypted relational databases. In *DASFAA*, 2004.
[14] B. Hore, S. Mehrotra, M. Canim, and M. Kantarcioglu. Secure multidimensional range queries over outsourced data. *VLDBJ*. To Appear.
[15] H. Hu, J. Xu, C. Ren, and B. Choi. Processing private queries over untrusted data cloud through privacy homomorphism. In *ICDE*, 2011.
[16] E. Mykletun and G. Tsudik. Aggregation queries in the database-as-a-service model. In *DBSec*, 2006.
[17] S. Papadopoulos, S. Bakiras, and D. Papadias. Nearest neighbor search with strong location privacy. *PVLDB*, 3(1):619–629, 2010.
[18] M. Sharifzadeh and C. Shahabi. VoR-Tree: R-trees with voronoi diagrams for efficient processing of spatial nearest neighbor queries. *PVLDB*, 2010.
[19] E. Shi, J. Bethencourt, H. T.-H. Chan, D. X. Song, and A. Perrig. Multi-dimensional range query over encrypted data. In *IEEE Symposium on Security and Privacy*, 2007.
[20] P. Williams and R. Sion. Usable pir. In *NDSS*, 2008.
[21] W. K. Wong, D. W.-L. Cheung, B. Kao, and N. Mamoulis. Secure knn computation on encrypted databases. In *SIGMOD*, 2009.
[22] B. Yao, F. Li, and X. Xiao. Secure nearest neighbor revisited. Technical report, http://www.cs.utah.edu/~lifeifei/papers/snn.pdf.