Secure Nearest Neighbor Revisited

Bin Yao\textsuperscript{1}, Feifei Li\textsuperscript{2}, Xiaokui Xiao\textsuperscript{3}

\textsuperscript{1}Department of Computer Science and Engineering
Shanghai Jiao Tong University, P. R. China

\textsuperscript{2}School of Computing
University of Utah

\textsuperscript{3}School of Computer Engineering
Nanyang Technological University, Singapore

July 11, 2013
The Motivation

- Cloud databases: Google Cloud SQL, Microsoft SQL Azure, Amazon SimpleDB.
Cloud databases: Google Cloud SQL, Microsoft SQL Azure, Amazon SimpleDB.
Service providers (SP) answer queries from different clients.
The Motivation

- Cloud databases: Google Cloud SQL, Microsoft SQL Azure, Amazon SimpleDB.
- Service providers (SP) answer queries from different clients.
- Data owner might not want to reveal data values to SP; clients might not want SP to learn their queries and/or the query results.

Hakan Hacigumus, Balakrishna R. Iyer, Chen Li, Sharad Mehrotra: Executing SQL over encrypted data in the database-service-provider model. SIGMOD 2002
Introduction and Motivation

Secure Query Processing

Secure Nearest Neighbor (SNN)

cloud server
Introduction and Motivation

Secure Query Processing

Secure Nearest Neighbor (SNN)

Bin Yao, Feifei Li, Xiaokui Xiao

Secure Nearest Neighbor Revisited
Secure Query Processing

Secure Nearest Neighbor (SNN)

cloud server

data owner

\( E(D) \)
Introduction and Motivation

Secure Query Processing

Secure Nearest Neighbor (SNN)

cloud server

data owner

client

\[ E(D) \]
Introduction and Motivation

Secure Query Processing

Secure Nearest Neighbor (SNN)

E(D)

cloud server

data owner

client
Secure Query Processing

Secure Nearest Neighbor (SNN)

E(D)

cloud server

data owner

client
Introduction and Motivation

Secure Query Processing

Secure Nearest Neighbor (SNN)

Bin Yao, Feifei Li, Xiaokui Xiao

Secure Nearest Neighbor Revisited
Introduction and Motivation

- Secure Query Processing

Secure Nearest Neighbor (SNN)

Cloud server

Data owner

Client

\[ E(D) \]
Introduction and Motivation

- Secure Query Processing
  - Secure Nearest Neighbor (SNN)

$E(D)$

cloud server

data owner

client
Introduction and Motivation

- Secure Query Processing
  - Secure Nearest Neighbor (SNN)

Bin Yao, Feifei Li, Xiaokui Xiao

Secure Nearest Neighbor Revisited
Introduction and Motivation

- Secure Query Processing
  - Secure Nearest Neighbor (SNN)

\[ E(D) \]

Data owner

Cloud server

Client

Bin Yao, Feifei Li, Xiaokui Xiao
Secure Nearest Neighbor Revisited
Related Work

Existing work has examined the problems of answering basic SQL queries [1], executing aggregate queries [2], and performing range queries [3], over an encrypted database. Hu et al. [4] and Wong et al. [5] deal with the SNN problem; the solutions thus proposed, however, are insecure and can be attacked efficiently.

Fully homomorphic encryption due to Craig Gentry, “A Fully Homomorphic Encryption Scheme (Ph.D. thesis)” is mostly of theoretical interest, impractical, and inefficient for large data.


Bin Yao, Feifei Li, Xiaokui Xiao
Secure Nearest Neighbor Revisited
Related Work

Existing work has examined the problems of answering basic SQL queries [1], executing aggregate queries [2], and performing range queries [3], over an encrypted database.


Related Work

- Existing work has examined the problems of answering basic SQL queries \[1\], executing aggregate queries \[2\], and performing range queries \[3\], over an encrypted database.
- Hu et al. \[4\] and Wong et al. \[5\] deal with the SNN problem; the solutions thus proposed, however, are insecure and can be attacked efficiently.


Existing work has examined the problems of answering basic SQL queries [1], executing aggregate queries [2], and performing range queries [3], over an encrypted database.

Hu et al. [4] and Wong et al. [5] deal with the SNN problem; the solutions thus proposed, however, are insecure and can be attacked efficiently.

Fully homomorphic encryption due to Craig Gentry, “A Fully Homomorphic Encryption Scheme (Ph.D. thesis)” : mostly of theoretical interest, impractical, and inefficient for large data.


Problem Formulation

Three parties:
- A data owner who has a database $D$ that contains $d$-dimensional Euclidean objects/points, and outsources $D$ to a server that cannot be fully trusted.
- A client (or multiple of them) who wants to access and pose queries to $D$.
- A server that is honest but potentially curious in the tuples in the database and the queries from the clients.

Objective:
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.
To ensure the SNN method is as secure as the encryption method $E$ used by the data owner.

Adversary model: same as whatever model in which $E$ is secure, e.g., IND-CPA, IND-CCA.
Problem Formulation

- Three parties:

  - A data owner who has a database $D$ that contains $d$-dimensional Euclidean objects/points, and outsources $D$ to a server that cannot be fully trusted.
  - A client (or multiple of them) who wants to access and pose queries to $D$.
  - A server that is honest but potentially curious in the tuples in the database and the queries from the clients.

Objective:
- 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.
- To ensure the SNN method is as secure as the encryption method $E$ used by the data owner.

Adversary model: same as whatever model in which $E$ is secure, e.g., IND-CPA, IND-CCA.
Problem Formulation

Three parties:

- A **data owner** who has a database $D$ that contains $d$-dimensional Euclidean objects/points, and outsources $D$ to a server that cannot be fully trusted.
- A **client** (or multiple of them) who wants to access and pose queries to $D$.
- A **server** that is **honest but potentially curious** in the tuples in the database and the queries from the clients.

Objective:

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.

To ensure the SNN method is as secure as the encryption method $E$ used by the data owner.

Adversary model: same as whatever model in which $E$ is secure, e.g., IND-CPA, IND-CCA.
Three parties:

- A **data owner** who has a database $D$ that contains $d$-dimensional Euclidean objects/points, and outsources $D$ to a server that cannot be fully trusted.
- A **client** (or multiple of them) who wants to access and pose queries to $D$.
- A **server** that is *honest but potentially curious* in the tuples in the database and the queries from the clients.

Objective:
Problem Formulation

- Three parties:
  - A *data owner* who has a database $D$ that contains $d$-dimensional Euclidean objects/points, and outsources $D$ to a server that cannot be fully trusted.
  - A *client* (or multiple of them) who wants to access and pose queries to $D$.
  - A *server* that is *honest but potentially curious* in the tuples in the database and the queries from the clients.

- Objective:
  - 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.
  - To ensure the SNN method is as secure as the encryption method $E$ used by the data owner.
Problem Formulation

- **Three parties:**
  - A *data owner* who has a database $D$ that contains $d$-dimensional Euclidean objects/points, and outsources $D$ to a server that cannot be fully trusted.
  - A *client* (or multiple of them) who wants to access and pose queries to $D$.
  - A *server* that is *honest but potentially curious* in the tuples in the database and the queries from the clients.

- **Objective:**
  - 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.
  - To ensure the SNN method is as secure as the encryption method $E$ used by the data owner.
  - Adversary model: same as whatever model in which $E$ is secure, e.g., IND-CPA, IND-CCA.
Database $D = \{p_1, \ldots, p_N\}$, where $p_i \in \mathbb{R}^d$. 

Goal: find a method $S$ such that $S(E(q), E(D)) = E(nn(q, D))$, where $q \in \mathbb{R}^d$, without letting the SP learn contents about either the query (and its results) or the tuples in $D$. 

Standard security model, such as indistinguishability under chosen plaintext attack (IND-CPA), or indistinguishability under chosen ciphertext attack (IND-CCA). 

To appear in ICDE'13.
Problem Definition

- Database $D = \{p_1, \ldots, p_N\}$, where $p_i \in \mathbb{R}^d$.
- $E(D)$: encryption of $D$ under a secure encryption function $E$. 

To appear in ICDE'13.

Bin Yao, Feifei Li, Xiaokui Xiao

Secure Nearest Neighbor Revisited
Problem Definition

- Database $D = \{p_1, \ldots, p_N\}$, where $p_i \in \mathbb{R}^d$.
- $E(D)$: encryption of $D$ under a secure encryption function $E$.
- Goal: find a method $S$ such that $S(E(q), E(D)) = E(nn(q, D))$, where $q \in \mathbb{R}^d$, without letting the SP learn contents about either the query (and its results) or the tuples in $D$. 

Standard security model, such as indistinguishability under chosen plaintext attack (IND-CPA), or indistinguishability under chosen ciphertext attack (IND-CCA). To appear in ICDE'13.
Problem Definition

- Database $D = \{p_1, \ldots, p_N\}$, where $p_i \in \mathbb{R}^d$.
- $E(D)$: encryption of $D$ under a secure encryption function $E$.
- Goal: find a method $S$ such that $S(E(q), E(D)) = E(\text{nn}(q, D))$, where $q \in \mathbb{R}^d$, without letting the SP learn contents about either the query (and its results) or the tuples in $D$.
- Standard security model, such as indistinguishability under chosen plaintext attack (IND-CPA), or indistinguishability under chosen ciphertext attack (IND-CCA).
Problem Definition

- Database $D = \{p_1, \ldots, p_N\}$, where $p_i \in \mathbb{R}^d$.
- $E(D)$: encryption of $D$ under a secure encryption function $E$.
- Goal: find a method $S$ such that $S(E(q), E(D)) = E(\text{nn}(q, D))$, where $q \in \mathbb{R}^d$, without letting the SP learn contents about either the query (and its results) or the tuples in $D$.
- Standard security model, such as indistinguishability under chosen plaintext attack (IND-CPA), or indistinguishability under chosen ciphertext attack (IND-CCA).
First attempt: Wai Kit Wong, David Wai-Lok Cheung, Ben Kao, Nikos Mamoulis: Secure kNN computation on encrypted databases. SIGMOD 2009
First attempt: Wai Kit Wong, David Wai-Lok Cheung, Ben Kao, Nikos Mamoulis: Secure kNN computation on encrypted databases. SIGMOD 2009

Basic idea: construct a “secure” encryption function that preserves the dot product between a query point and a database point.
First attempt: Wai Kit Wong, David Wai-Lok Cheung, Ben Kao, Nikos Mamoulis: Secure kNN computation on encrypted databases. SIGMOD 2009

- Basic idea: construct a “secure” encryption function that preserves the dot product between a query point and a database point.
- Attack we found: after learning only $d$ query points and their encryptions, a linear system of $d$ equations can be formed to decrypt any encrypted $p \in D$. 
Second attempt: Haibo Hu, Jianliang Xu, Chushi Ren, Byron Choi: Processing private queries over untrusted data cloud through privacy homomorphism. ICDE 2011
Insecurity of Existing Methods

- Second attempt: Haibo Hu, Jianliang Xu, Chushi Ren, Byron Choi: Processing private queries over untrusted data cloud through privacy homomorphism. ICDE 2011
  - Basic idea: Using homomorphic encryption to encrypt each entry in a multi-dimensional index; Guide the search by using the homomorphic operations between (encrypted) $q$ and entry $e$. 

Secure Nearest Neighbor Revisited
Insecurity of Existing Methods

- **Second attempt:** Haibo Hu, Jianliang Xu, Chushi Ren, Byron Choi: Processing private queries over untrusted data cloud through privacy homomorphism. ICDE 2011
  - Basic idea: Using homomorphic encryption to encrypt each entry in a multi-dimensional index; Guide the search by using the homomorphic operations between (encrypted) $q$ and entry $e$.
  - Attack we found: In the above process, the server learns if $q$ lies to the left or the right of another point, in each dimension, which leads to a binary search to efficiently recover any encrypted point.
Order-preserving encryption (OPE) is a set of functions \(\{E, E^{-1}, op\}\), such that:
- \(E(m) = c, E^{-1}(c) = m\) (here we omit the keys).
- \(op(c_1, c_2) = 1\) if \(m_1 < m_2\); \(op(c_1, c_2) = -1\) if \(m_1 > m_2\).
Order-preserving encryption (OPE) is a set of functions \( \{\mathcal{E}, \mathcal{E}^{-1}, op\} \), such that:

- \( \mathcal{E}(m) = c, \mathcal{E}^{-1}(c) = m \) (here we omit the keys).
- \( op(c_1, c_2) = 1 \) if \( m_1 < m_2 \); \( op(c_1, c_2) = -1 \) if \( m_1 > m_2 \).

**Theorem**

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 (IND-OCPA).

Rakesh Agrawal, Jerry Kiernan, Ramakrishnan Srikant, Yirong Xu: Order-Preserving Encryption for Numeric Data. SIGMOD 2004
Alexandra Boldyreva, Nathan Chenette, Younho Lee, Adam O’Neill: Order-Preserving Symmetric Encryption. EUROCRYPT 2009
Given $E(D) = \{E(p_1), \ldots, E(p_N)\}$, suppose we have a secure SNN method $S$ such that: $S(E(q), E(D)) \rightarrow E(nn(q, D))$ without the knowledge of $E^{-1}$.
Hardness of the Problem: SNN gives OPE

- Given $E(D) = \{E(p_1), \ldots, E(p_N)\}$, suppose we have a secure SNN method $S$ such that: $S(E(q), E(D)) \rightarrow E(nn(q, D))$ without the knowledge of $E^{-1}$.
- We can construct an OPE, $\{E, E^{-1}, op\}$, based on $S(\cdot)$!
Hardness of the Problem: SNN gives OPE

- Given \( E(D) = \{ E(p_1), \ldots, E(p_N) \} \), suppose we have a secure SNN method \( S \) such that: \( S(E(q), E(D)) \rightarrow E(nn(q, D)) \) without the knowledge of \( E^{-1} \).
- We can construct an OPE, \( \{ \mathcal{E}, \mathcal{E}^{-1}, \text{op} \} \), based on \( S(\cdot) \):

\[
h(\cdot) : p_{i+1} - p_i < p_i - p_{i-1}
\]

\[
\begin{array}{cccccc}
m_1 & m_2 & m_3 & m_4 \\
p_1 & p_2 & p_3 & p_4 & p_5 & Z^+
\end{array}
\]
Hardness of the Problem: SNN gives OPE

- Given $E(D) = \{E(p_1), \ldots, E(p_N)\}$, suppose we have a secure SNN method $S$ such that: $S(E(q), E(D)) \rightarrow E(nn(q, D))$ without the knowledge of $E^{-1}$.
- We can construct an OPE, $\{\mathcal{E}, \mathcal{E}^{-1}, op\}$, based on $S(\cdot)$!

\[
\mathcal{h}(\cdot): p_{i+1} - p_i < p_i - p_{i-1}
\]

\[
\mathcal{E}(m_i) = E(h(m_i)) = E(p_i), \ \mathcal{E}^{-1}(c) = h^{-1}(E^{-1}(c))
\]
Given $E(D) = \{E(p_1), \ldots, E(p_N)\}$, suppose we have a secure SNN method $S$ such that: $S(E(q), E(D)) \rightarrow E(nn(q, D))$ without the knowledge of $E^{-1}$.

We can construct an OPE, $\{E, E^{-1}, op\}$, based on $S(\cdot)$!

$$h(\cdot): p_{i+1} - p_i < p_i - p_{i-1}$$

$$\mathbb{Z}^+$$

$D$: $p_1 \ p_2 \ p_3 \ p_4 \ p_5$

$E(m_i) = E(h(m_i) = E(p_i), E^{-1}(c) = h^{-1}(E^{-1}(c))$

$nn(p_i, D) = p_{i+1}, \ i \in [1, N]; \ nn(p_{N+1}, D) = p_N.$
Hardness of the Problem: SNN gives OPE

- Given $E(D) = \{E(p_1), \ldots, E(p_N)\}$, suppose we have a secure SNN method $S$ such that: $S(E(q), E(D)) \rightarrow E(nn(q, D))$ without the knowledge of $E^{-1}$.
- We can construct an OPE, $\{\mathcal{E}, \mathcal{E}^{-1}, op\}$, based on $S(\cdot)$!

\[ h(\cdot) : p_{i+1} - p_i < p_i - p_{i-1} \]

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

\[ nn(p_i, D) = p_{i+1}, \quad i \in [1, N]; \quad nn(p_{N+1}, D) = p_N. \]

\[ S(E(p_i), E(D)) = E(p_{i+1}), \quad \text{for } i \in [1, N]. \]

\[ S(E(p_{N+1}), E(D)) = E(p_N). \]
How to construct $op(\mathcal{E}(m_i), \mathcal{E}(m_j))$?
Hardness of the Problem: SNN gives OPE

- How to construct $op(\mathcal{E}(m_i), \mathcal{E}(m_j))$?
- Observe that by our construction, $\mathcal{E}(m_i) = E(p_i)$, and $\mathcal{E}(m_j) = E(p_j)$.
How to construct \( op(\mathcal{E}(m_i), \mathcal{E}(m_j)) \)?

Observe that by our construction, \( \mathcal{E}(m_i) = E(p_i) \), and \( \mathcal{E}(m_j) = E(p_j) \).

Define function \( \text{traverse}(\mathcal{E}(m_i)) \) which outputs \( i \)! 
How to construct $o_{\mathcal{P}}(\mathcal{E}(m_i), \mathcal{E}(m_j))$?

Observe that by our construction, $\mathcal{E}(m_i) = E(p_i)$, and $\mathcal{E}(m_j) = E(p_j)$.

Define function $\text{traverse}(\mathcal{E}(m_i))$ which outputs $i$!

$$h(\cdot) : p_{i+1} - p_i < p_i - p_{i-1}$$

Bin Yao, Feifei Li, Xiaokui Xiao
Secure Nearest Neighbor Revisited
Hardness of the Problem: SNN gives OPE

- How to construct $op(E(m_i), E(m_j))$?
- Observe that by our construction, $E(m_i) = E(p_i)$, and $E(m_j) = E(p_j)$.
- Define function $\text{traverse}(E(m_i))$ which outputs $i$!

$$h(\cdot): p_{i+1} - p_i < p_i - p_{i-1}$$

1: $S(E(p_2), E(D) = E(p_3)$
How to construct $\text{op}(\mathcal{E}(m_i), \mathcal{E}(m_j))$?

Observe that by our construction, $\mathcal{E}(m_i) = E(p_i)$, and $\mathcal{E}(m_j) = E(p_j)$.

Define function $\text{traverse}(\mathcal{E}(m_i))$ which outputs $i!$

Define $h(\cdot): p_{i+1} - p_i < p_i - p_{i-1}$

![Diagram](image-url)
How to construct $\text{op}(\mathcal{E}(m_i), \mathcal{E}(m_j))$?

- Observe that by our construction, $\mathcal{E}(m_i) = E(p_i)$, and $\mathcal{E}(m_j) = E(p_j)$.

- Define function $\text{traverse}(\mathcal{E}(m_i))$ which outputs $i$!

$$h(\cdot) : p_{i+1} - p_i < p_i - p_{i-1}$$

2: $S(E(p_3), E(D) = E(p_4)$
How to construct $\text{op}(\mathcal{E}(m_i), \mathcal{E}(m_j))$?

Observe that by our construction, $\mathcal{E}(m_i) = E(p_i)$, and $\mathcal{E}(m_j) = E(p_j)$.

Define function $\text{traverse}(\mathcal{E}(m_i))$ which outputs $i$!

$$h(\cdot) : p_{i+1} - p_i < p_i - p_{i-1}$$
How to construct $o_p(\mathcal{E}(m_i), \mathcal{E}(m_j))$?

Observe that by our construction, $\mathcal{E}(m_i) = E(p_i)$, and $\mathcal{E}(m_j) = E(p_j)$.

Define function $\text{traverse}(\mathcal{E}(m_i))$ which outputs $i!$

$$h(\cdot) : p_{i+1} - p_i < p_i - p_{i-1}$$

$$m_1 \quad m_2 \quad m_3 \quad m_4$$

$$\mathcal{E}(m_2)$$

$$p_1 \quad p_2 \quad p_3 \quad p_4 \quad p_5$$

$$E(p_2) \quad E(p_3) \quad E(p_4)$$

3: $S(E(p_4), E(D) = E(p_5)$
How to construct $\text{op}(E(m_i), E(m_j))$?

Observe that by our construction, $E(m_i) = E(p_i)$, and $E(m_j) = E(p_j)$.

Define function $\text{traverse}(E(m_i))$ which outputs $i!$

$$h(\cdot): p_{i+1} - p_i < p_i - p_{i-1}$$
Hardness of the Problem: SNN gives OPE

- How to construct \( op(\mathcal{E}(m_i), \mathcal{E}(m_j)) \)?
- Observe that by our construction, \( \mathcal{E}(m_i) = E(p_i) \), and \( \mathcal{E}(m_j) = E(p_j) \).
- Define function \( \text{traverse}(\mathcal{E}(m_i)) \) which outputs \( i! \)

\[
\mathcal{E}(m_i) = E(p_i) \quad \Rightarrow \quad \mathcal{E}(m_j) = E(p_j) \quad \Rightarrow \quad \text{traverse}(\mathcal{E}(m_i)) \text{ outputs } i!
\]

\[
h(\cdot) : p_{i+1} - p_i < p_i - p_{i-1}
\]

\[
m_1 \quad m_2 \quad m_3 \quad m_4
\]

\[
\mathcal{E}(m_2) \quad \mathcal{E}(m_3) \quad \mathcal{E}(m_4)
\]

4: \( S(E(p_5), E(D) = E(p_4), \text{Repetition FOUND!} \)
How to construct $\text{op}(\mathcal{E}(m_i), \mathcal{E}(m_j))$?

Observe that by our construction, $\mathcal{E}(m_i) = E(p_i)$, and $\mathcal{E}(m_j) = E(p_j)$.

Define function $\text{traverse}(\mathcal{E}(m_i))$ which outputs $i$!

$$h(\cdot) : p_{i+1} - p_i < p_i - p_{i-1}$$

Define function $\text{traverse}(\mathcal{E}(m_i))$ which outputs $i$!

$$h(\cdot) : p_{i+1} - p_i < p_i - p_{i-1}$$

$$\mathcal{E}(m_1) \mathcal{E}(m_2) \mathcal{E}(m_3) \mathcal{E}(m_4)$$

$$E(p_1) E(p_2) E(p_3) E(p_4) E(p_5)$$

4: $S(E(p_5), E(D) = E(p_4)$, Repetition FOUND!

$i = N - (\text{number of steps} - 2)!$
It only says it is hard to output $E(\text{nn}(q, D))$! What if we relax this restriction and allow something “less precise”?
It only says it is hard to output $E(\text{nn}(q, D))$! What if we relax this restriction and allow something “less precise”? Extreme case: just return $E(D)$ and ask client to decrypt and find $\text{nn}(q, D)$. Obviously secure! But expensive!

The SVD (secure voronoi diagram) method:

Create partitions based on the voronoi cells of $D$.

$E(D) = \{E(G_1), E(G_2), \ldots\}$. Send partition configurations (the boundaries) to clients, client only needs to ask for the encryption of a given partition by partition id (which is figured out locally).
So, Hopeless? NO!

- It only says it is hard to output $E(\text{nn}(q, D))$! What if we relax this restriction and allow something “less precise”?
- Extreme case: just return $E(D)$ and ask client to decrypt and find $\text{nn}(q, D)$. Obviously secure! But expensive!
- The SVD (secure voronoi diagram) method:
So, Hopeless? NO!

- It only says it is hard to output $E(\text{nn}(q, D))$! What if we relax this restriction and allow something “less precise”?
- Extreme case: just return $E(D)$ and ask client to decrypt and find $\text{nn}(q, D)$. Obviously secure! But expensive!
- The SVD (secure voronoi diagram) method:
  - create partitions based on the voronoi cells of $D$. 

$$G_i = \{ p \mid p \subset B_i \}$$

![Diagram of voronoi cells and partitions](image)
So, Hopeless? NO!

- It only says it is hard to output $E(\text{nn}(q, D))$! What if we relax this restriction and allow something “less precise”?
- Extreme case: just return $E(D)$ and ask client to decrypt and find $\text{nn}(q, D)$. Obviously secure! But expensive!
- The SVD (secure voronoi diagram) method:
  - create partitions based on the voronoi cells of $D$.
  - $E(D) = \{E(G_1), E(G_2), \ldots \}$.

$$\text{G}_i = \{p \mid p \subset B_i\}$$
So, Hopeless? NO!

- It only says it is hard to output $E(\text{nn}(q, D))$! What if we relax this restriction and allow something “less precise”?
- Extreme case: just return $E(D)$ and ask client to decrypt and find $\text{nn}(q, D)$. Obviously secure! But expensive!
- The SVD (secure voronoi diagram) method:
  - create partitions based on the voronoi cells of $D$.
  - $E(D) = \{E(G_1), E(G_2), \ldots\}$.
  - send partition configurations (the boundaries) to clients, client only needs to ask for the encryption of a given partition by partition id (which is figured out locally).

$$G_i = \{p | p \subset B_i\}$$
So, Hopeless? NO!

- It only says it is hard to output $E(\text{nn}(q, D))$! What if we relax this restriction and allow something “less precise”?
- Extreme case: just return $E(D)$ and ask client to decrypt and find $\text{nn}(q, D)$. Obviously secure! But expensive!
- The SVD (secure voronoi diagram) method:
  - create partitions based on the voronoi cells of $D$.
  - $E(D) = \{E(G_1), E(G_2), \ldots \}$.
  - send partition configurations (the boundaries) to clients, client only needs to ask for the encryption of a given partition by partition id (which is figured out locally).

$$G_i = \{p | p \subset B_i\}$$

Challenge:
$$\text{minmax}(|G_i|)!$$
Solution Overview

- Secure Voronoi Diagram (SVD):
  - Preprocessing at the data owner
  - Query processing at the client
Solution Overview

- Secure Voronoi Diagram (SVD):
  - Preprocessing at the data owner
  - Query processing at the client
Preprocessing at the data owner:
Preprocessing at the data owner:
Preprocessing at the data owner:

- Data owner

\[ D \xrightarrow{\text{partitioning}} G(D) \]

Discussed later...
Preprocessing at the data owner:

Data owner

\[ D \rightarrow \text{partitioning} \rightarrow G(D) \]
Solution Overview

- Preprocessing at the data owner:

  D \xrightarrow{\text{partitioning}} G(D)

  D \xrightarrow{\text{random padding operation}} G'(D)

  data owner
Solution Overview

- Preprocessing at the data owner:

  \[ D \rightarrow G(D) \]
  
  \[ G'(D) \rightarrow \text{encryption} \rightarrow E(G'(D)) \rightarrow \text{outsourcing} \rightarrow \text{cloud server} \]

  random padding operation

  partitioning

  discussed later...
Preprocessing at the data owner:

- **D**: Data set
- **G(D)**: Transformation of data set
- **G'(D)**: Random padding operation on **D**
- **E(G'(D))**: Encryption of padded data
- **E^{-1}, P(D)**: Decryption and privacy preservation
- **G(D)**: Transformed data set

Diagram:
- Data owner
- Cloud server
- Client
- Partitioning discussed later...
Secure Voronoi Diagram (SVD):

- Preprocessing at the data owner
- Query processing at the client
Query processing at the client:
Query processing at the client:
Query processing at the client:

\[ P(D) \]

\[ \begin{array}{c}
0 & 1 \\
2 & 3
\end{array} \]

\[ q \]

\[ i = 1 \]
Query processing at the client:

- Compute the ID(index): $i = 1$
- Encryption: $E(i) = 100...010...$
Solution Overview

- Query processing at the client:

Client: \( P(D) \)

Query: \( q \)

- Compute the ID(index): \( i = 1 \)

Encryption: \( E(i) = 100...010... \)

Cloud Server: Query
Query processing at the client:

1. Client queries the cloud server with a encrypted query $E(q)$.
2. The cloud server computes the ID (index) $i$ for the query.
3. The client receives the encrypted ID $E(i)$.
4. The client decrypts the ID to $i = 1$.

Note: The diagram illustrates the process with a 2D grid, where the client queries a point $q$ and the cloud server returns the index $i = 1$. The encryption and decryption processes are indicated by arrows.
Query processing at the client:

- **P(D)**
- **q**
- **E(i)** = 100...010...
- **G_i**
- **i = 1**
- **E(G_i)**
- **compute the ID(index)**
- **encryption**
- **decryption**

Bin Yao, Feifei Li, Xiaokui Xiao
Secure Nearest Neighbor Revisited
Query processing at the client:

1. The client sends a query to the cloud server.
2. The cloud server encrypts the data and sends it back to the client.
3. The client decrypts the data and finds the nearest neighbor in the P(D) space.
4. The client computes the ID (index) of the nearest neighbor.
5. The client queries the cloud server with the encrypted ID (E(i) = 100...010...).

Diagram:
- Client
- Cloud server
- Encryption (E(G_i))
- Decryption
- Find the NN
- Compute the ID (index)
- Query

Equation:
\[ E(i) = 100...010... \]
SVD Partitioning Principle

1. $B_i$ is an axis-parallel d-dimensional box and $B_i \cap B_j = \emptyset$ for any $i \neq j$.

2. $G_i = \{ p_j | v_c(j) \text{ is contained or intersected by } B_i \}$.

3. The minimum $|G_x|$ and minimum $|G_x| - |G_i|$, which means low storage and communication overheads, as well as cheap encryption cost.
SVD Partitioning Principle

**Bin Yao, Feifei Li, Xiaokui Xiao**

Secure Nearest Neighbor Revisited
Bin Yao, Feifei Li, Xiaokui Xiao
Secure Nearest Neighbor Revisited
SVD Partitioning Principle

$B_i$ is an axis-parallel $d$-dimensional box and $B_i \cap B_j = \emptyset$ for any $i \neq j$.

$G_i = \{p_j | \text{vc}_j \text{ is contained or intersected by } B_i\}$

minimum $|G_x|$ and minimum $|G_x| - |G_i|$, which means low storage and communication overheads, as well as cheap encryption cost.
SVD Partitioning Principle

$B_i$ is an axis-parallel $d$-dimensional box and $B_i \cap B_j = \emptyset$ for any $i \neq j$.

$G_i$ is a subset of dataset $D$.

$B_i$ is the geometric boundary of $G_i$.

The diagram shows a partitioning of the dataset $D$ into subsets $G_1$ and $G_2$, with geometric boundaries $B_1$ and $B_2$. Points $p_1$ to $p_{16}$ are marked within and outside the subsets and boundaries.
**SVD Partitioning Principle**

$G(D)$

$G_i$: a subset of dataset $D$

$B_i$: the geometric boundary of $G_i$

$G_i = \{ p_j | v_c(j) \text{ is contained or intersected by } B_i \}$

$B_i$ is an axis-parallel $d$-dimensional box and $B_i \cap B_j = \emptyset$ for any $i \neq j$
SVD Partitioning Principle

$G(D)$

$B_1$

$B_2$

$G_1$

$G_2$

$G_i$: a subset of dataset $D$

$B_i$: the geometric boundary of $G_i$

1. $B_i$ is an axis-parallel $d$-dimensional box and $B_i \cap B_j = \emptyset$ for any $i \neq j$

2. $G_i = \{p_j|vc_j$ is contained or intersected by $B_i\}$
SVD Partitioning Principle

\[ G(D) \]

\[ B_1 \]
\[ p_1 \]
\[ p_2 \]
\[ p_3 \]
\[ p_4 \]
\[ p_5 \]
\[ p_6 \]
\[ p_7 \]
\[ p_8 \]
\[ p_9 \]
\[ p_{10} \]
\[ p_{11} \]
\[ p_{12} \]
\[ p_{13} \]
\[ p_{14} \]
\[ p_{15} \]
\[ p_{16} \]

\[ B_2 \]

\[ G_1 \]
\[ G_2 \]

\[ G_1 = \{p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8, p_{10}\} \]
\[ G_2 = \{p_5, p_6, p_7, p_8, p_{10}, p_9, p_{11}, p_{12}, p_{13}, p_{14}, p_{15}, p_{16}\} \]

1. \( B_i \) is an axis-parallel \( d \)-dimensional box and \( B_i \cap B_j = \emptyset \) for any \( i \neq j \)
2. \( G_i = \{p_j | v_{c_j} \text{ is contained or intersected by } B_i\} \)
G_i : a subset of dataset D
B_i : the geometric boundary of G_i
|G_i|: size of G_i

1. B_i is an axis-parallel d-dimensional box and B_i ∩ B_j = ∅ for any i ≠ j
2. G_i = \{p_j|vc_j is contained or intersected by B_i\}
**SVD Partitioning Principle**

- $G_i$ : a subset of dataset $D$
- $B_i$ : the geometric boundary of $G_i$
- $|G_i|$ : size of $G_i$
  - $|G_1| = 9$
  - $|G_2| = 12$

**Diagram:**

- $G_1 = \{p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8, p_{10}\}$
- $G_2 = \{p_5, p_6, p_7, p_8, p_{10}, p_9, p_{11}, p_{12}, p_{13}, p_{14}, p_{15}, p_{16}\}$

1. $B_i$ is an axis-parallel $d$-dimensional box and $B_i \cap B_j = \emptyset$ for any $i \neq j$
2. $G_i = \{p_j | \text{vc}_j \text{ is contained or intersected by } B_i\}$
SVD Partitioning Principle

\[ G(D) \]

\[ B_1 \]

\[ B_2 \]

\[ G_1 \]

\[ G_2 \]

\[ G_i : \text{a subset of dataset } D \]

\[ B_i : \text{the geometric boundary of } G_i \]

\[ |G_i| : \text{size of } G_i \]

\[ |G_1| = 9 \]

\[ |G_2| = 12 \]

\[ G_x : \text{largest-sized partition} \]

1. \( B_i \) is an axis-parallel \( d \)-dimensional box and \( B_i \cap B_j = \emptyset \) for any \( i \neq j \)

2. \( G_i = \{p_j | \text{vc}_j \text{ is contained or intersected by } B_i\} \)
SVD Partitioning Principle

$G_i$: a subset of dataset $D$

$B_i$: the geometric boundary of $G_i$

$|G_i|$: size of $G_i$

$|G_1| = 9$

$|G_2| = 12$

$G_x$: largest-sized partition

$\begin{align*}
G_1 &= \{p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8, p_{10}\} \\
G_2 &= \{p_5, p_6, p_7, p_8, p_{10}, p_9, p_{11}, p_{12}, p_{13}, p_{14}, p_{15}, p_{16}\}
\end{align*}$

1. $B_i$ is an axis-parallel $d$-dimensional box and $B_i \cap B_j = \emptyset$ for any $i \neq j$

2. $G_i = \{p_j \mid v_{c_j} \text{ is contained or intersected by } B_i\}$
SVD Partitioning Principle

$G(D)$

$G_1$ : a subset of dataset $D$

$B_i$ : the geometric boundary of $G_i$

$|G_i|$: size of $G_i$

$|G_1| = 9$

$|G_2| = 12$

$G_x$: largest-sized partition

$G_1 = \{p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8, p_{10}\}$

$G_2 = \{p_5, p_6, p_7, p_8, p_{10}, p_9, p_{11}, p_{12}, p_{13}, p_{14}, p_{15}, p_{16}\}$

1. $B_i$ is an axis-parallel $d$-dimensional box and $B_i \cap B_j = \emptyset$ for any $i \neq j$

2. $G_i = \{p_j | \text{vc}_j \text{ is contained or intersected by } B_i\}$

3. minimum $|G_x|$ and minimum $|G_x| - |G_i|$, which means low storage and communication overheads, as well as cheap encryption cost
SVD Partitioning Principle

$G_i$ : a subset of dataset $D$

$B_i$ : the geometric boundary of $G_i$

$|G_i|$ : size of $G_i$

$G_x$ : largest-sized partition

Duplicated points

$G_1 = \{p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8, p_9\}$

$G_2 = \{p_5, p_6, p_7, p_8, p_{10}, p_9, p_{11}, p_{12}, p_{13}, p_{14}, p_{15}, p_{16}\}$

1. $B_i$ is an axis-parallel $d$-dimensional box and $B_i \cap B_j = \emptyset$ for any $i \neq j$

2. $G_i = \{p_j | v_{c_j}$ is contained or intersected by $B_i\}$

3. minimum $|G_x|$ and minimum $|G_x| - |G_i|$, which means low storage and communication overheads, as well as cheap encryption cost
SVD Partitioning

- Square Grid (SG)
- Minimum Space Grid (MinSG)
- Minimum Maximum Partition (MinMax)
SVD Partitioning

- Square Grid (SG)
- Minimum Space Grid (MinSG)
- Minimum Maximum Partition (MinMax)
Square Grid (SG)

$B_2 = C_{1,2}$  $B_4 = C_{2,2}$

$B_1 = C_{1,1}$  $B_3 = C_{2,1}$

G(D)
Square Grid (SG)

- **Merits:**

- **Demerits:**
Square Grid (SG)

- **Merits:**
  - simple
  - minimum storage cost at client

- **Demerits:**

Bin Yao, Feifei Li, Xiaokui Xiao
Secure Nearest Neighbor Revisited
Square Grid (SG)

- **Merits:**
  - simple
  - minimum storage cost at client

- **Demerits:**
  - high storage and communication overheads, as well as expensive encryption cost because of highly unbalanced partitions when the data distribution is skewed
SVD Partitioning

- Square Grid (SG)
- Minimum Space Grid (MinSG)
- Minimum Maximum Partition (MinMax)
Minimum Space Grid (MinSG)
Minimum Space Grid (MinSG)

\[ |G| = 26 \]
Minimum Space Grid (MinSG)

- A greedy algorithm: always split the maximum partition $G_x$ into smaller partitions
Minimum Space Grid (MinSG)

- A greedy algorithm: always split the maximum partition $G_x$ into smaller partitions
- use a line going through the entire space and intersected with the voronoi vertex in $B_x$
Minimum Space Grid (MinSG)

A greedy algorithm: always split the maximum partition $G_x$ into smaller partitions

use a line going through the entire space and intersected with the voronoi vertex in $B_x$
A greedy algorithm: always split the maximum partition $G_x$ into smaller partitions

use a line going through the entire space and intersected with the voronoi vertex in $B_x$
Minimum Space Grid (MinSG)

- A greedy algorithm: always split the maximum partition $G_x$ into smaller partitions
- Use a line going through the entire space and intersected with the Voronoi vertex in $B_x$
A greedy algorithm: always split the maximum partition $G_x$ into smaller partitions

use a line going though the entire space and intersected with the voronoi vertex in $B_x$
A greedy algorithm: always split the maximum partition $G_x$ into smaller partitions

use a line going though the entire space and intersected with the voronoi vertex in $B_x$
A greedy algorithm: always split the maximum partition $G_x$ into smaller partitions

use a line going though the entire space and intersected with the voronoi vertex in $B_x$

choose the $\ell$ that leads to the minimum maximum partition
A greedy algorithm: always split the maximum partition $G_x$ into smaller partitions

use a line going though the entire space and intersected with the voronoi vertex in $B_x$

choose the $\ell$ that leads to the minimum maximum partition
A greedy algorithm: always split the maximum partition $G_x$ into smaller partitions

use a line going though the entire space and intersected with the voronoi vertex in $B_x$

choose the $\ell$ that leads to the minimum maximum partition
A greedy algorithm: always split the maximum partition $G_x$ into smaller partitions

- use a line going though the entire space and intersected with the voronoi vertex in $B_x$
- choose the $\ell$ that leads to the minimum maximum partition
A greedy algorithm: always split the maximum partition $G_x$ into smaller partitions.

use a line going through the entire space and intersected with the voronoi vertex in $B_x$.

choose the $\ell$ that leads to the minimum maximum partition.
A greedy algorithm: always split the maximum partition $G_x$ into smaller partitions.

- use a line going though the entire space and intersected with the voronoi vertex in $B_x$
- choose the $\ell$ that leads to the minimum maximum partition
A greedy algorithm: always split the maximum partition $G_x$ into smaller partitions.

use a line going though the entire space and intersected with the voronoi vertex in $B_x$

choose the $\ell$ that leads to the minimum maximum partition
A greedy algorithm: always split the maximum partition $G_x$ into smaller partitions

use a line going though the entire space and intersected with the voronoi vertex in $B_x$

choose the $\ell$ that leads to the minimum maximum partition
A greedy algorithm: always split the maximum partition $G_x$ into smaller partitions

- use a line going though the entire space and intersected with the voronoi vertex in $B_x$
- choose the $\ell$ that leads to the minimum maximum partition
A greedy algorithm: always split the maximum partition $G_x$ into smaller partitions

use a line going though the entire space and intersected with the voronoi vertex in $B_x$

choose the $\ell$ that leads to the minimum maximum partition
A greedy algorithm: always split the maximum partition $G_x$ into smaller partitions.

- use a line going through the entire space and intersected with the voronoi vertex in $B_x$
- choose the $\ell$ that leads to the minimum maximum partition
Minimum Space Grid (MinSG)

- **Merits:**
  - relatively balanced partitions: low storage and communication overheads, as well as cheap encryption cost

- **Demerits:**
  - complicated partitioning process
  - not most balanced: small-sized partitions introduced by some unnecessary splitting

Bin Yao, Feifei Li, Xiaokui Xiao

Secure Nearest Neighbor Revisited
Minimum Space Grid (MinSG)

- Merits:
  - relatively balanced partitions: low storage and communication overheads, as well as cheap encryption cost

- Demerits:
Minimum Space Grid (MinSG)

- **Merits:**
  - relatively balanced partitions: low storage and communication overheads, as well as cheap encryption cost

- **Demerits:**
  - complicated partitioning process
  - not most balanced: small-sized partitions introduced by some unnecessary splitting
Minimum Space Grid (MinSG)

$|G_{11}| = 11$
$|G_{12}| = 10$
$|G_{21}| = 14$
$|G_{22}| = 6$
Minimum Space Grid (MinSG)

\[ |G_{11}| = 11 \]
\[ |G_{12}| = 10 \]
\[ |G_{21}| = 14 \]
\[ |G_{22}| = 6 \]
Minimum Space Grid (MinSG)
Minimum Space Grid (MinSG)

We need a method that produce more balanced partitions!!
SVD Partitioning

- Square Grid (SG)
- Minimum Space Grid (MinSG)
- Minimum Maximum Partition (MinMax)
Minimum Maximum Partition (MinMax)

$G$
Minimum Maximum Partition (MinMax)

$|G| = 26$

- similar to MinSG in most part
Minimum Maximum Partition (MinMax)

$G$

| $G$ | $= 26$ |

similar to MinSG in most part
Minimum Maximum Partition (MinMax)

- similar to MinSG in most part
- use **segments** going though the space bounded by $B_x$ instead of lines going though the entire space to split partitions
similar to MinSG in most part

use **segments** going though the space bounded by $B_x$ instead of lines going though the entire space to split partitions
Minimum Maximum Partition (MinMax)

similar to MinSG in most part

use **segments** going though the space bounded by $B_x$ instead of lines going though the entire space to split partitions
Minimum Maximum Partition (MinMax)

$|G| = 26$

similar to MinSG in most part

use **segments** going though the space bounded by $B_x$ instead of lines going though the entire space to split partitions
Minimum Maximum Partition (MinMax)

- similar to MinSG in most part
- use **segments** going though the space bounded by $B_x$ instead of lines going though the entire space to split partitions
Minimum Maximum Partition (MinMax)

- similar to MinSG in most part
- use **segments** going though the space bounded by $B_x$ instead of lines going though the entire space to split partitions

$|G| = 26$
similar to MinSG in most part
use **segments** going though the space bounded by $B_x$ instead of lines going though the entire space to split partitions
similar to MinSG in most part

use **segments** going through the space bounded by $B_x$ instead of lines going through the entire space to split partitions
similar to MinSG in most part

use segments going though the space bounded by $B_x$ instead of lines going though the entire space to split partitions
Minimum Maximum Partition (MinMax)

- similar to MinSG in most part
- use **segments** going though the space bounded by $B_x$ instead of lines going though the entire space to split partitions
similar to MinSG in most part

use **segments** going though the space bounded by $B_x$ instead of lines going though the entire space to split partitions
Minimum Maximum Partition (MinMax)

- similar to MinSG in most part
- use **segments** going though the space bounded by $B_x$ instead of lines going though the entire space to split partitions
Minimum Maximum Partition (MinMax)

- similar to MinSG in most part
- use segments going though the space bounded by $B_x$ instead of lines going though the entire space to split partitions
Minimum Maximum Partition (MinMax)

- similar to MinSG in most part
- use **segments** going though the space bounded by $B_x$ instead of lines going though the entire space to split partitions

$G_1$ \hspace{2cm} $|G_1| = 16$

$G_2$ \hspace{2cm} $|G_2| = 15$

$B_1$ \hspace{2cm} $B_2$
similar to MinSG in most part

use **segments** going though the space bounded by $B_x$ instead of lines going though the entire space to split partitions
similar to MinSG in most part
use segments going though the space bounded by $B_x$ instead of lines going though the entire space to split partitions
similar to MinSG in most part

use **segments** going though the space bounded by $B_x$ instead of lines going though the entire space to split partitions
Similar to MinSG in most part

- use **segments** going though the space bounded by $B_x$ instead of lines going though the entire space to split partitions
similar to MinSG in most part

- use **segments** going though the space bounded by \( B_x \) instead of lines going though the entire space to split partitions
Minimum Maximum Partition (MinMax)

- similar to MinSG in most part
- use segments going though the space bounded by $B_x$ instead of lines going though the entire space to split partitions
Minimum Maximum Partition (MinMax)

- **Merits:**
  - most balanced partitions: low storage and communication overheads, as well as cheap encryption cost

- **Demerits:**
  - high storage cost at client
Merits:

- most balanced partitions: low storage and communication overheads, as well as cheap encryption cost

Demerits:
Minimum Maximum Partition (MinMax)

- **Merits:**
  - most balanced partitions: low storage and communication overheads, as well as cheap encryption cost

- **Demerits:**
  - high storage cost at client
Comparison between MinSG and MinMax

Clearly, MinMax achieves more balanced partitions than MinSG, which means lower storage and communication overheads, as well as cheaper encryption cost.
Comparison between MinSG and MinMax

Clearly, MinMax achieves more balanced partitions than MinSG, which means lower storage and communication overheads, as well as cheaper encryption cost.

Bin Yao, Feifei Li, Xiaokui Xiao
Secure Nearest Neighbor Revisited
Comparison between MinSG and MinMax

MinSG

MinMax

- Clearly, MinMax achieves more balanced partitions than MinSG, which means lower storage and communication overheads, as well as cheaper encryption cost.
We examine the three methods: SG, MinSG and MinMax. For each method, we test its running time of both partition phrase and encryption phrase, partition size, communication cost of both the preprocessing step and query step and query time.

C++, Linux, Intel Xeon 3.07GHz CPU and 8GB memory.

Data sets: 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 $D_{\text{max}}$ and form smaller datasets based on $D_{\text{max}}$.

Default settings.

Symbol | Definition | Default Value
---|---|---
$D$ | size of the dataset | $10^6$
$k$ | number of partitions | 625
$DT$ | dataset type | CA

Bin Yao, Feifei Li, Xiaokui Xiao
Secure Nearest Neighbor Revisited
We examine the three methods: SG, MinSG and MinMax.
We examine the three methods: SG, MinSG and MinMax.

For each method, we test its running time of both partition phrase and encryption phrase, partition size, communication cost of both the preprocessing step and query step and query time.

Data sets

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 $D_{\text{max}}$ and form smaller datasets based on $D_{\text{max}}$.

Default settings.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Definition</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$</td>
<td>D</td>
<td>\text{size of the dataset}$</td>
</tr>
<tr>
<td>$k$</td>
<td>number of partitions</td>
<td>625</td>
</tr>
<tr>
<td>$DT$</td>
<td>dataset type</td>
<td>CA</td>
</tr>
</tbody>
</table>

Bin Yao, Feifei Li, Xiaokui Xiao

Secure Nearest Neighbor Revisited
We examine the three methods: SG, MinSG and MinMax.

For each method, we test its running time of both partition phrase and encryption phrase, partition size, communication cost of both the preprocessing step and query step and query time.

C++, Linux, Intel Xeon 3.07GHz CPU and 8GB memory
We examine the three methods: SG, MinSG and MinMax.

For each method, we test its running time of both partition phrase and encryption phrase, partition size, communication cost of both the preprocessing step and query step and query time.

C++, Linux, Intel Xeon 3.07GHz CPU and 8GB memory

Data sets

- 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 $D_{\text{max}}$ and form smaller datasets based on $D_{\text{max}}$. 
We examine the three methods: SG, MinSG and MinMax.

For each method, we test its running time of both partition phrase and encryption phrase, partition size, communication cost of both the preprocessing step and query step and query time.

C++, Linux, Intel Xeon 3.07GHz CPU and 8GB memory

Data sets

- 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 $D_{\text{max}}$ and form smaller datasets based on $D_{\text{max}}$.

Default settings.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Definition</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$</td>
<td>D</td>
<td>$</td>
</tr>
<tr>
<td>$k$</td>
<td>number of partitions</td>
<td>625</td>
</tr>
<tr>
<td>$DT$</td>
<td>dataset type</td>
<td>CA</td>
</tr>
</tbody>
</table>
- Vary $|D|$: Wai Kit Wong, David Cheung, Ben Kao, Nikos Mamoulis:

Secure kNN computation on encrypted databases. SIGMOD 2009

![Graph showing time to attack vs $|D|=N \times 10^6$]
Attack on Existing SNN Methods

- **Vary $|D|$**: Haibo Hu, Jianliang Xu, Chushi Ren, Byron Choi: Processing private queries over untrusted data cloud through privacy homomorphism. ICDE 2011
Partition size in different methods

- Vary $k$

![Graph showing partition size in different methods. The x-axis represents $k$ with values 0, 300, 600, 900, and 1200. The y-axis represents the average partition size (bytes) in logarithmic scale. The graph compares three methods: SG, MinSG, and MinMax. Each method is represented by different markers and error bars, indicating the variation in partition size for each value of $k$.](image)
Partition size in different methods

- Vary $|D|$
Query communication cost

- Vary $k$

![Graph showing query communication cost with varying $k$ values.](image)
Query communication cost

- Vary $|D|$
Vary $k$

Total running time of the preprocessing step

- $SG$
- $MinSG$
- $MinMax$
- $Send-D$

$k$ vs. total running time (secs)

Logarithmic scale on the y-axis

$k$: 0, 300, 600, 900, 1200

$SG$, $MinSG$, $MinMax$, $Send-D$ graphs plotted
Vary $|D|$
Query time for different methods

- Vary $k$
Query time for different methods

- Vary $|D|$
Running time of the partition phase

- Vary $k$

```
   SG  MinSG  MinMax
```

- Bin Yao, Feifei Li, Xiaokui Xiao
- Secure Nearest Neighbor Revisited
Running time of the partition phase

- Vary $|D|$
- Vary $k$

![Graph showing total size of $E(D)$]

- $\text{SG}$
- $\text{MinSG}$
- $\text{MinMax}$

- $\text{Send-D} \times |D|$

- Bin Yao, Feifei Li, Xiaokui Xiao
- Secure Nearest Neighbor Revisited
Vary $|D|$
Open Problems

1. Other similarity metrics?
Open Problems

1. Other similarity metrics?
2. High dimensions (beyond 2)?
Open Problems

1. Other similarity metrics?
2. High dimensions (beyond 2)?
3. Secure $k$ nearest neighbors?
Open Problems

1. Other similarity metrics?
2. High dimensions (beyond 2)?
3. Secure $k$ nearest neighbors?
4. Updates?

Secure data analytics based on similarity search: clustering, content-based search, etc.

Variants of similarity search: reverse nearest neighbors, skylines, etc.

Bin Yao, Feifei Li, Xiaokui Xiao
Secure Nearest Neighbor Revisited
Open Problems

1. Other similarity metrics?
2. High dimensions (beyond 2)?
3. Secure $k$ nearest neighbors?
4. Updates?
5. Secure data analytics based on similarity search: clustering, content-based search, etc.
Open Problems

1. Other similarity metrics?
2. High dimensions (beyond 2)?
3. Secure $k$ nearest neighbors?
4. Updates?
5. Secure data analytics based on similarity search: clustering, content-based search, etc.
6. Variants of similarity search: reverse nearest neighbors, skylines, etc.
Conclusion

- Design a new partition-based secure voronoi diagram (SVD) method.
Design a new partition-based secure voronoi diagram (SVD) method.

Implement the SVD with three partitioning methods.
Design a new partition-based secure voronoi diagram (SVD) method.

Implement the SVD with three partitioning methods.

Future work
  - extending our investigation to higher dimensions, $k$ nearest neighbors
Thank You

Q and A