Secure Data Analytics

Feifei Li

School of Computing
University of Utah

September 21, 2014

The Motivation

- Cloud databases: Google Cloud SQL, Microsoft SQL Azure, Amazon SimpleDB.

Cloud Database

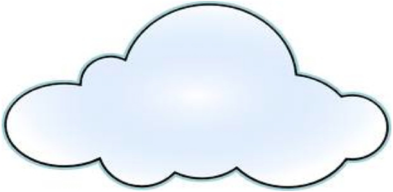
The Motivation

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

Cloud Database

The Motivation

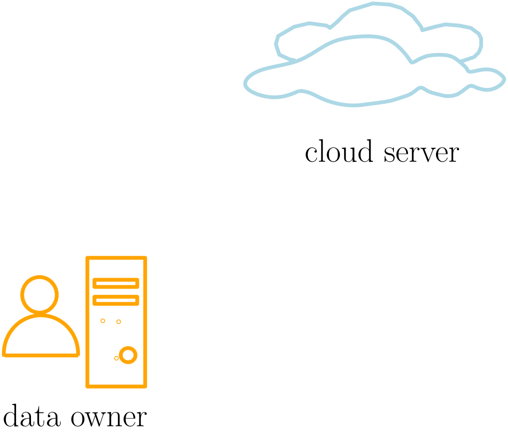
- Cloud databases: Google Cloud SQL, Microsoft SQL Azure, Amazon SimpleDB.
- Service providers (SP) answer queries from different clients.
- Data owner might not want to reveal data values to SP; clients might not want SP to learn their queries and/or the query results.



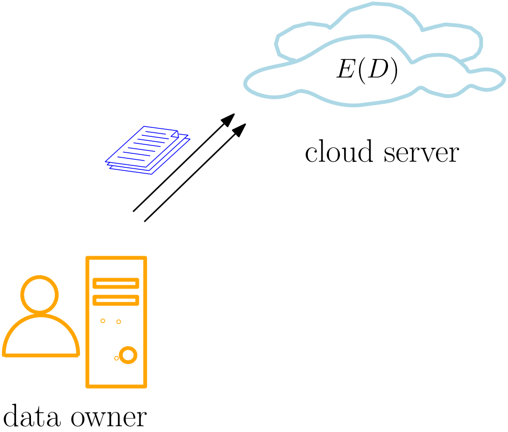
Cloud Database

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

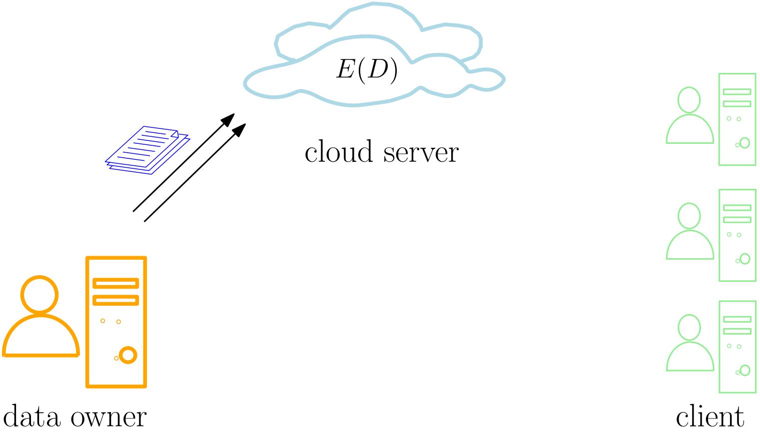
Introduction and Motivation



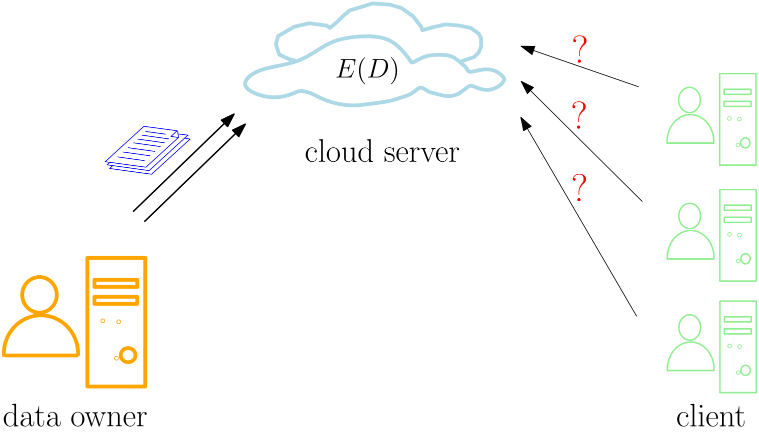
Introduction and Motivation



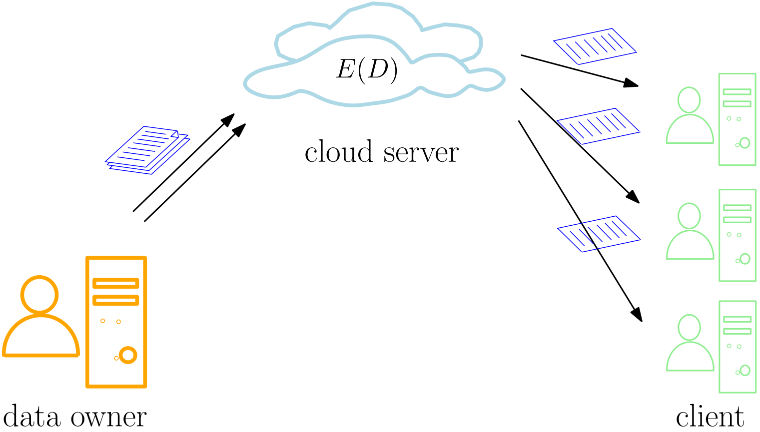
Introduction and Motivation



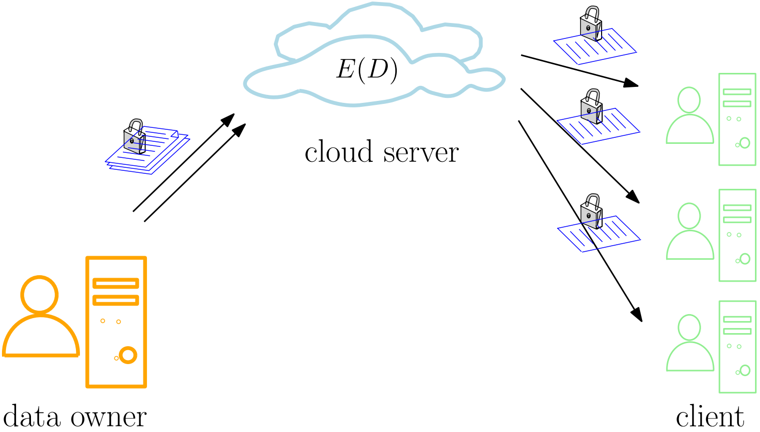
Introduction and Motivation



Introduction and Motivation

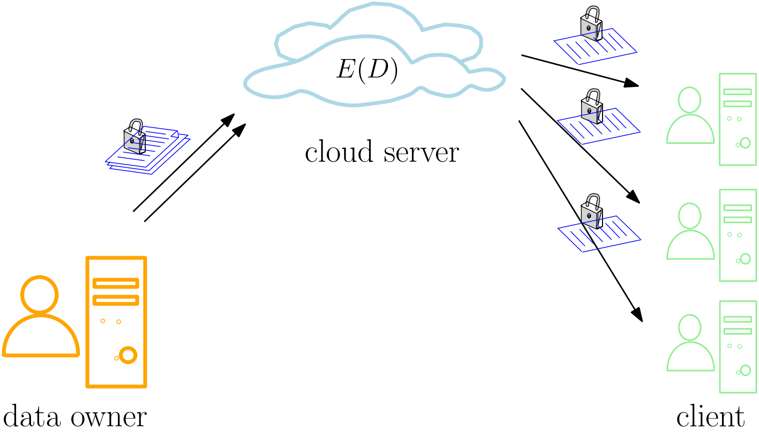


Introduction and Motivation



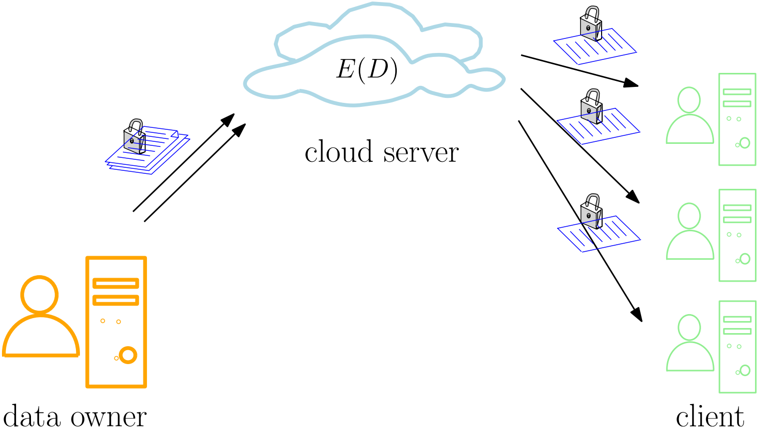
Introduction and Motivation

- Secure Query Processing



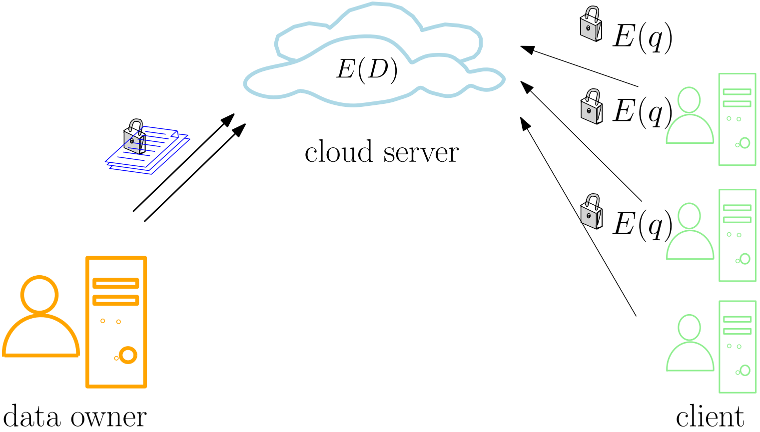
Introduction and Motivation

- Secure Query Processing
  - Secure Nearest Neighbor (SNN)

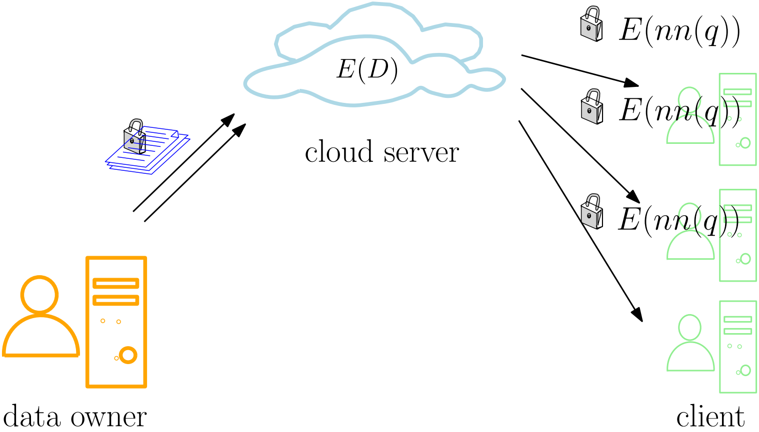


Introduction and Motivation

- Secure Query Processing
  - Secure Nearest Neighbor (SNN)



- Secure Query Processing
  - Secure Nearest Neighbor (SNN)



Related Work

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

- [1] H. Hacigumus, B. R. Iyer, C. Li, and S. Mehrotra. Executing SQL over encrypted data in the database service provider model. In SIGMOD, 2002
- [2] E. Mykletun and G. Tsudik. Aggregation queries in the database-as-a-service model. In DBSec, 2006
- [3] 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, pages 350C364, 2007

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

- [1] H. Hacigumus, B. R. Iyer, C. Li, and S. Mehrotra. Executing SQL over encrypted data in the database service provider model. In SIGMOD, 2002
- [2] E. Mykletun and G. Tsudik. Aggregation queries in the database-as-a-service model. In DBSec, 2006
- [3] 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, pages 350C364, 2007
- [4] H. Hu, J. Xu, C. Ren, and B. Choi. Processing private queries over untrusted data cloud through privacy homomorphism. In ICDE, pages 601C612, 2011
- [5] W. K. Wong, D. W.-L. Cheung, B. Kao, and N. Mamoulis. Secure knn computation on encrypted databases. In SIGMOD, pages 139C152, 2009

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

- [1] H. Hacigumus, B. R. Iyer, C. Li, and S. Mehrotra. Executing SQL over encrypted data in the database service provider model. In SIGMOD, 2002
- [2] E. Mykletun and G. Tsudik. Aggregation queries in the database-as-a-service model. In DBSec, 2006
- [3] 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, pages 350C364, 2007
- [4] H. Hu, J. Xu, C. Ren, and B. Choi. Processing private queries over untrusted data cloud through privacy homomorphism. In ICDE, pages 601C612, 2011
- [5] W. K. Wong, D. W.-L. Cheung, B. Kao, and N. Mamoulis. Secure knn computation on encrypted databases. In SIGMOD, pages 139C152, 2009

Problem Formulation

Problem Formulation

- Three parties:

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.

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:

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.

Problem Definition

- Database $D = \{p_1, \ldots, p_N\}$, where $p_i \in \mathbb{R}^d$.

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.

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.

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 $S(E(q), E(D)) = E(\mathrm{nn}(q, D))$, 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).

Insecurity of Existing Methods

- First attempt: Wai Kit Wong, David Wai-Lok Cheung, Ben Kao, Nikos Mamoulis: Secure kNN computation on encrypted databases. SIGMOD 2009

Insecurity of Existing Methods

- 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.

Insecurity of Existing Methods

- 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 $p \in D$.

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

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.

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.

Hardness of the Problem: OPE

- 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 $m_1 > m_2$.

Rakesh Agrawal, Jerry Kiernan, Ramakrishnan Srikant, Yirong Xu: Order-Preserving Encryption for Numeric Data. SIGMOD 2004

Hardness of the Problem: OPE

- 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
Alexandra Boldyreva, Nathan Chenette, Adam O'Neill: Order-Preserving Encryption Revisited: Improved Security Analysis and Alternative Solutions. CRYPTO 2011

Hardness of the Problem: SNN gives OPE

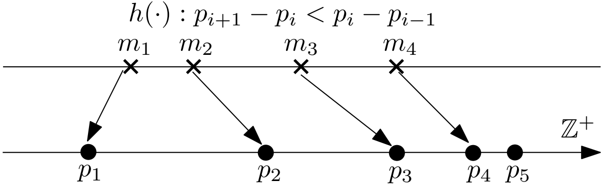
- Given $E(D) = \{E(p_1), \ldots, E(p_N)\}$, suppose we have a secure SNN method S such that: $S(E(q), E(D)) \rightarrow E(nn(q, D))$ without the knowledge of E^{-1}.

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 $S(\cdot)$!

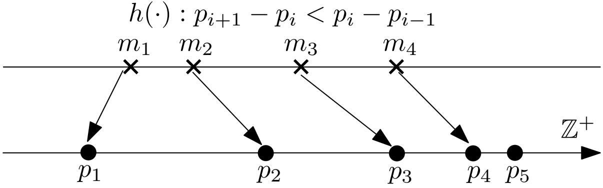
Hardness of the Problem: SNN gives OPE

- 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 $S(\cdot)$!



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 $S(\cdot)$!

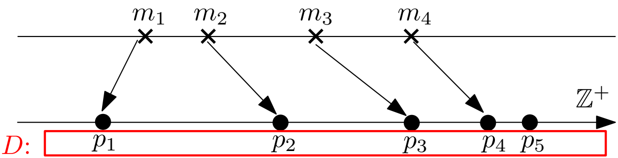


$\mathcal{E}(m_i) = E(h(m_i) = E(p_i)$, $\mathcal{E}^{-1}(c) = h^{-1}(E^{-1}(c))$

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 $S(\cdot)$!

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



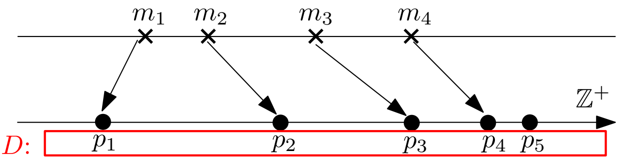
$$\mathcal{E}(m_i) = E(h(m_i) = E(p_i), \mathcal{E}^{-1}(c) = h^{-1}(E^{-1}(c))$$

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

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: $S(E(q), E(D)) \to E(nn(q, D))$ without the knowledge of E^{-1}.
- We can construct an OPE, $\{\mathcal{E}, \mathcal{E}^{-1}, op\}$, based on $S(\cdot)$!

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



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

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

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

$$S(E(p_{N+1}), E(D)) = E(p_N).$$

Hardness of the Problem: SNN gives OPE

- 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 $\mathcal{E}(m_j) = E(p_j)$.

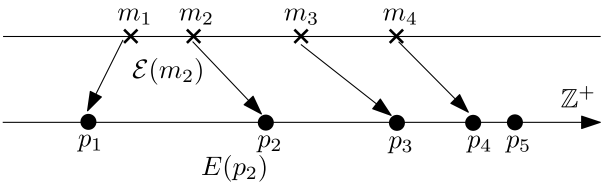
Hardness of the Problem: SNN gives OPE

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

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

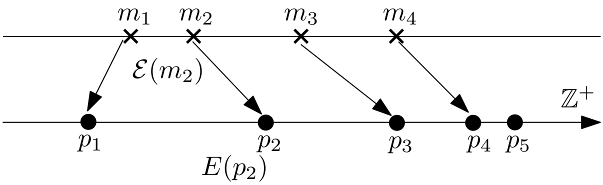
$$h(\cdot) : p_{i+1} - p_i < p_i - p_{i-1}$$



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

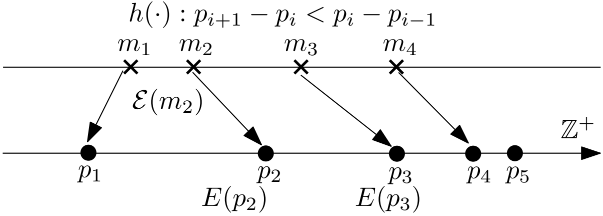
$h(\cdot) : p_{i+1} - p_i < p_i - p_{i-1}$



1: $S(E(p_2), E(D) = E(p_3)$

Hardness of the Problem: SNN gives OPE

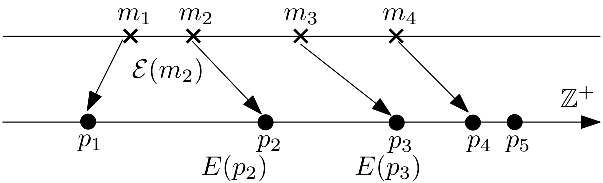
- How to construct $op(\mathcal{E}(m_i), \mathcal{E}(m_j))$?
- Observe that by our construction, $\mathcal{E}(m_i) = E(p_i)$, and $\mathcal{E}(m_j) = E(p_j)$.
- Define function traverse($\mathcal{E}(m_i)$) which outputs i!



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

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

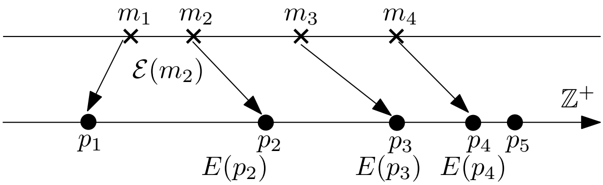


2: $S(E(p_3), E(D) = E(p_4)$

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

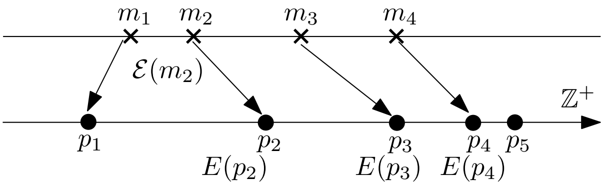
$h(\cdot) : p_{i+1} - p_i < p_i - p_{i-1}$



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

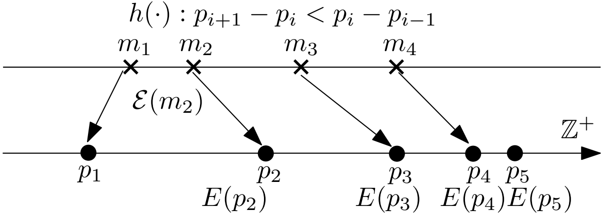
$h(\cdot) : p_{i+1} - p_i < p_i - p_{i-1}$



3: $S(E(p_4), E(D) = E(p_5)$

Hardness of the Problem: SNN gives OPE

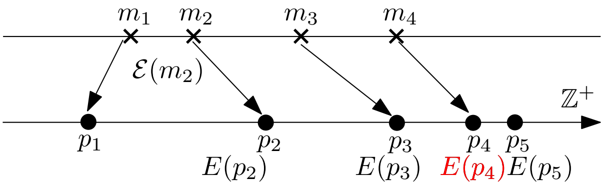
- How to construct $op(\mathcal{E}(m_i), \mathcal{E}(m_j))$?
- Observe that by our construction, $\mathcal{E}(m_i) = E(p_i)$, and $\mathcal{E}(m_j) = E(p_j)$.
- Define function traverse($\mathcal{E}(m_i)$) which outputs i!



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

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

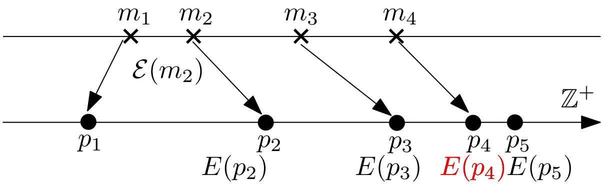


4: $S(E(p_5), E(D) = E(p_4)$, Repetition FOUND!

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

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



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

$i = N-$ (number of steps -2)!

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"?

So, Hopeless? NO!

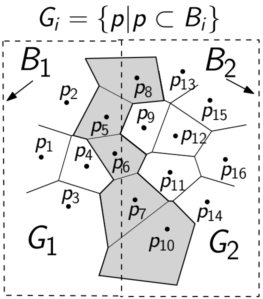
- It only says it is hard to output $E(\mathrm{nn}(q, D))$! What if we relax this restriction and allow something "less precise"?
- Extreme case: just return $E(D)$ and ask client to decrypt and find $\mathrm{nn}(q, D)$. Obviously secure! But expensive!

So, Hopeless? NO!

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

So, Hopeless? NO!

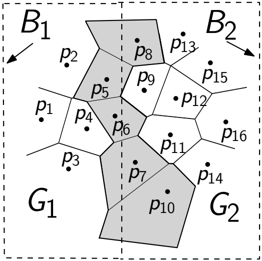
- It only says it is hard to output $E(\text{nn}(q, D))$! What if we relax this restriction and allow something "less precise"?
- Extreme case: just return $E(D)$ and ask client to decrypt and find $\text{nn}(q, D)$. Obviously secure! But expensive!
- The SVD (secure voronoi diagram) method:
  - create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



So, Hopeless? NO!

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

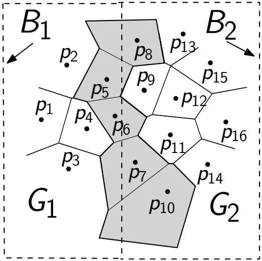
$$G_i = \{p | p \subset B_i\}$$



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).

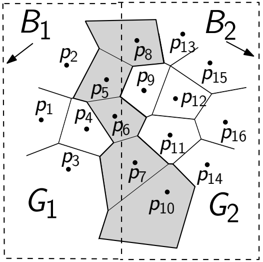
$$G_i = \{p | p \subset B_i\}$$



So, Hopeless? NO!

- It only says it is hard to output $E(\mathrm{nn}(q, D))$! What if we relax this restriction and allow something "less precise"?
- Extreme case: just return $E(D)$ and ask client to decrypt and find $\mathrm{nn}(q, D)$. 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: $\mathrm{minmax}(|G_i|)$!

Solution Overview

Solution Overview

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

Solution Overview

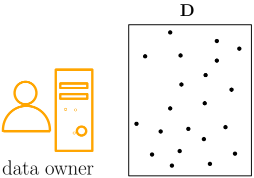
- Secure Voronoi Diagram (SVD):
  - Preprocessing at the data owner
  - Query processing at the client

Solution Overview

- Preprocessing at the data owner:

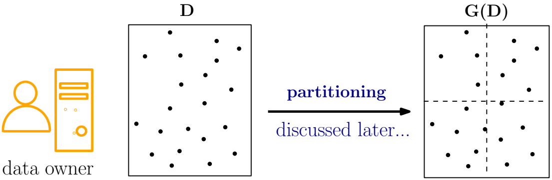
Solution Overview

- Preprocessing at the data owner:



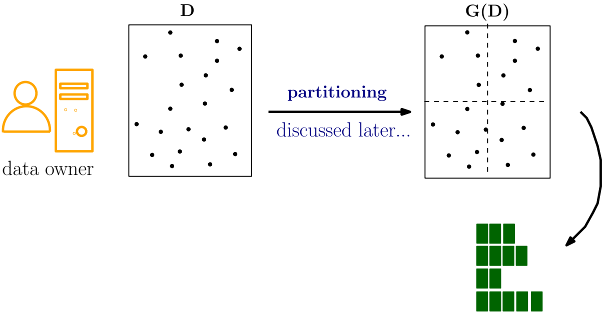
Solution Overview

- Preprocessing at the data owner:



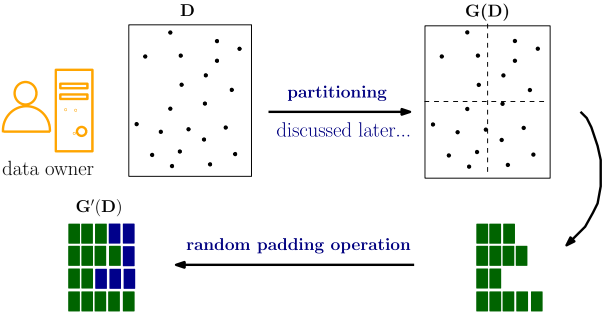
Solution Overview

- Preprocessing at the data owner:



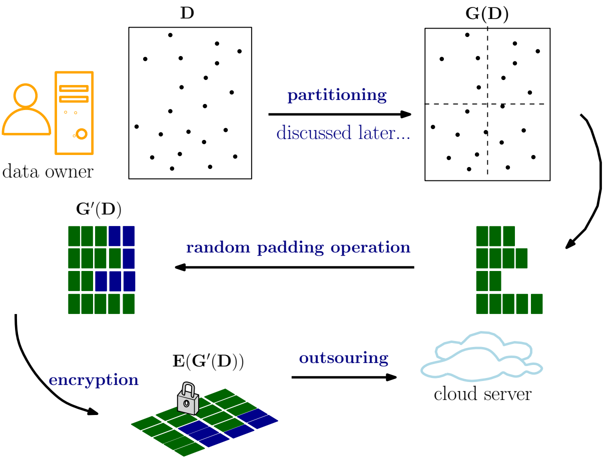
Solution Overview

- Preprocessing at the data owner:



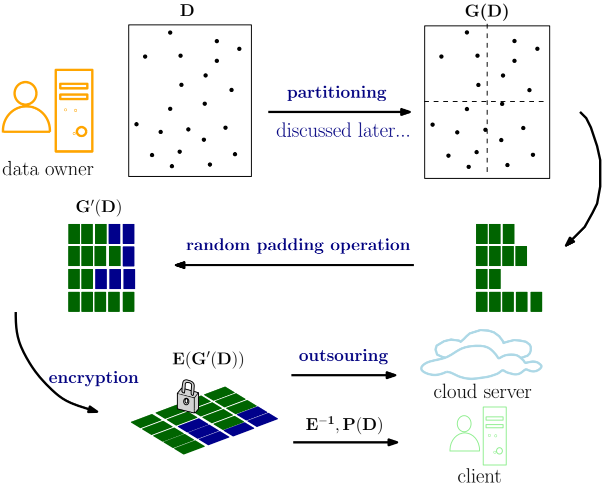
Solution Overview

- Preprocessing at the data owner:



Solution Overview

- Preprocessing at the data owner:



Solution Overview

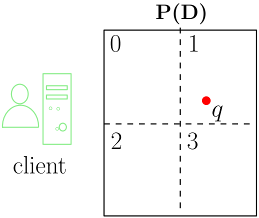
- Secure Voronoi Diagram (SVD):
  - Preprocessing at the data owner
  - Query processing at the client

Solution Overview

- Query processing at the client:

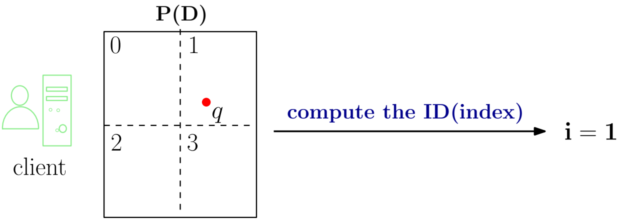
Solution Overview

- Query processing at the client:



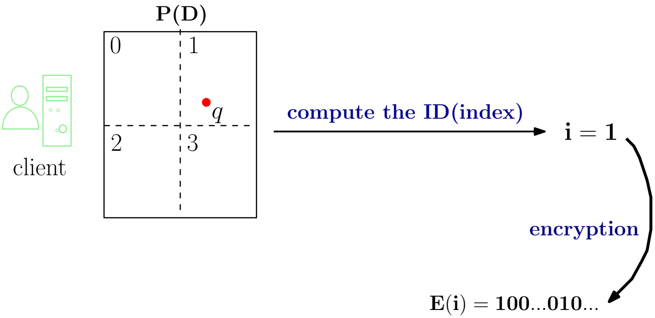
Solution Overview

- Query processing at the client:



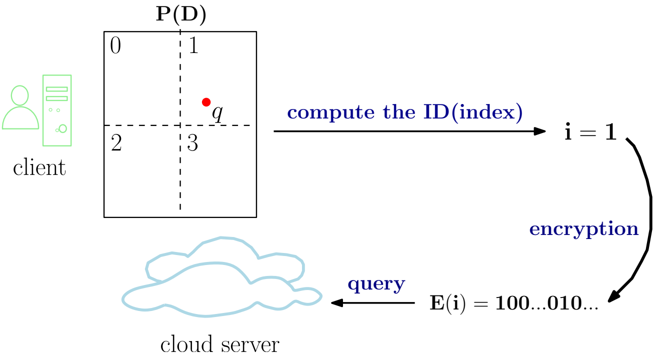
Solution Overview

- Query processing at the client:



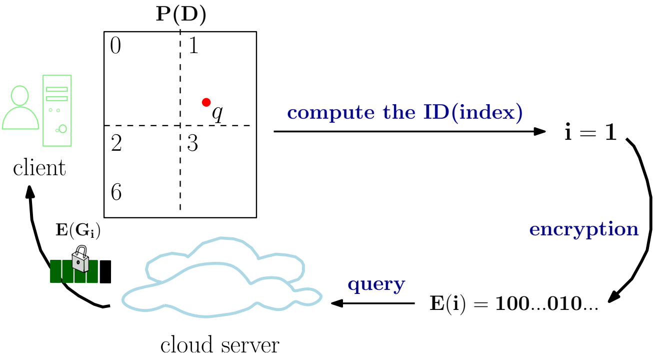
Solution Overview

- Query processing at the client:



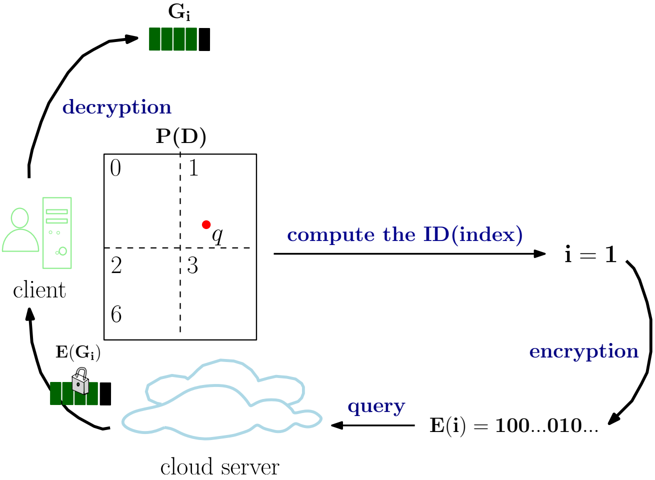
Solution Overview

- Query processing at the client:



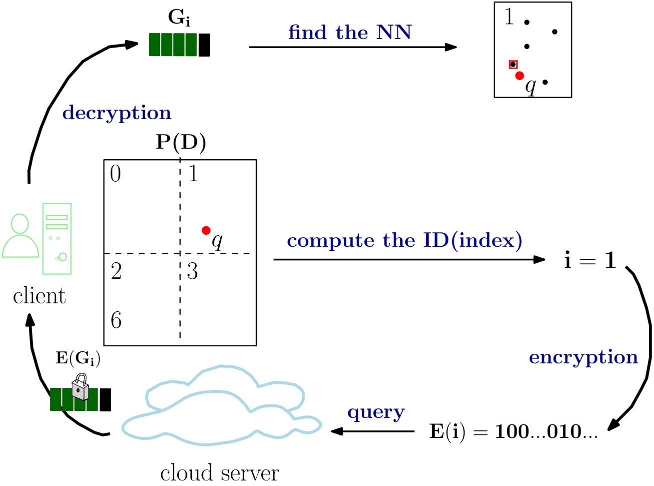
Solution Overview

- Query processing at the client:



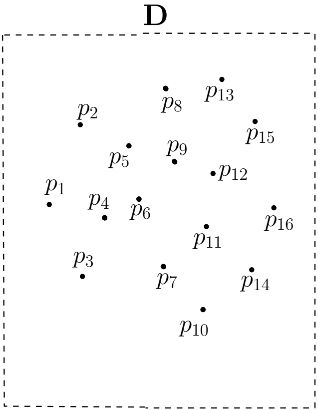
Solution Overview

- Query processing at the client:

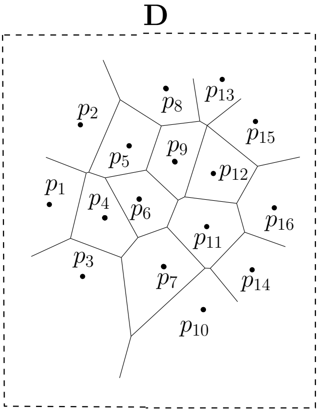


SVD Partitioning Principle

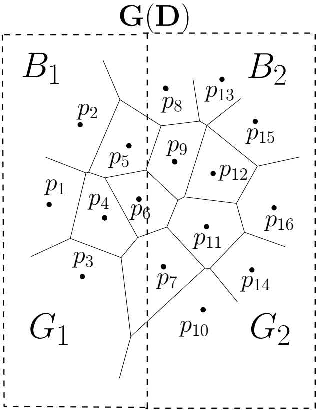
SVD Partitioning Principle



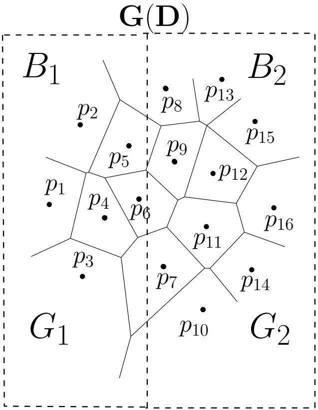
SVD Partitioning Principle



SVD Partitioning Principle



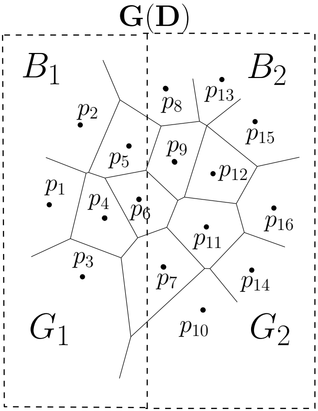
SVD Partitioning Principle



G_i : a subset of dataset D

B_i : the geometric boundary of G_i

SVD Partitioning Principle

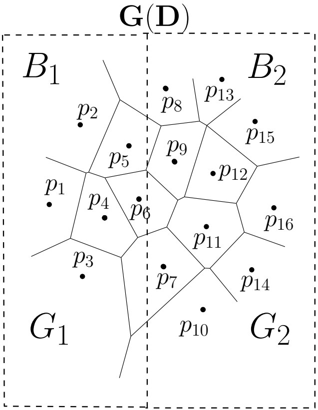


G_i : a subset of dataset D

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 $i \neq j$

SVD Partitioning Principle

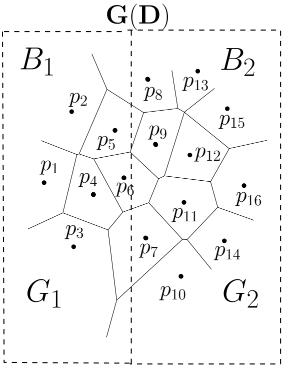


G_i : a subset of dataset D

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

SVD Partitioning Principle



G_i : a subset of dataset D

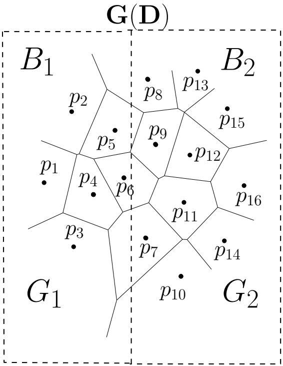
B_i : the geometric boundary of G_i

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

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

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

SVD Partitioning Principle



G_i : a subset of dataset D

B_i : the geometric boundary of G_i

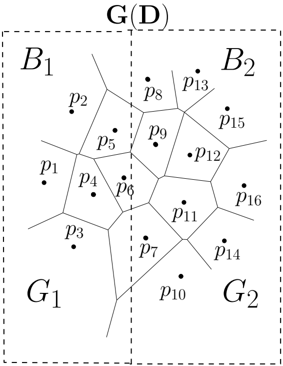
$|G_i|$: size of G_i

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

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

1. B_i is an axis-parallel d-dimensional box and $B_i \cap B_j = \emptyset$ for any $i \neq j$
2. $G_i = \{p_j | vc_j$ 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

$|G_1| = 9$

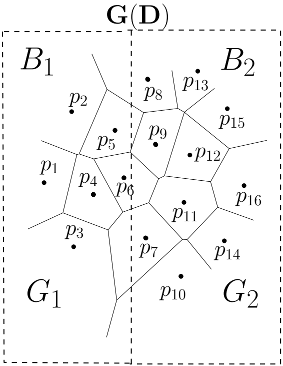
$|G_2| = 12$

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

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

1. B_i is an axis-parallel d-dimensional box and $B_i \cap B_j = \emptyset$ for any $i \neq j$
2. $G_i = \{p_j | vc_j$ 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

$|G_1| = 9$

$|G_2| = 12$

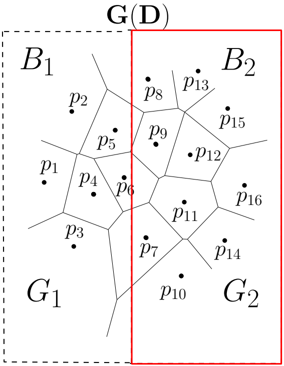
G_x: largest-sized partition

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

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

1. B_i is an axis-parallel d-dimensional box and $B_i \cap B_j = \emptyset$ for any $i \neq j$
2. $G_i = \{p_j | vc_j$ 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

$|G_1| = 9$

$|G_2| = 12$

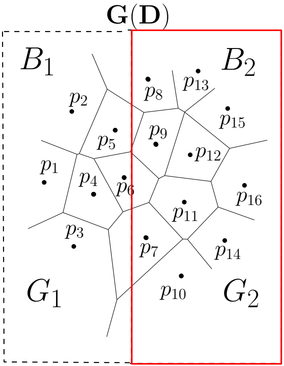
G_x: largest-sized partition

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

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

1. B_i is an axis-parallel d-dimensional box and $B_i \cap B_j = \emptyset$ for any $i \neq j$
2. $G_i = \{p_j | vc_j$ 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

$|G_1| = 9$

$|G_2| = 12$

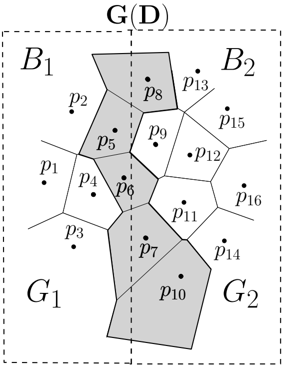
G_x: largest-sized partition

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

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

1. B_i is an axis-parallel d-dimensional box and $B_i \cap B_j = \emptyset$ for any $i \neq j$
2. $G_i = \{p_j | vc_j$ 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 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

Duplicated points

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

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

1. B_i is an axis-parallel d-dimensional box and $B_i \cap B_j = \emptyset$ for any $i \neq j$
2. $G_i = \{p_j | vc_j$ 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

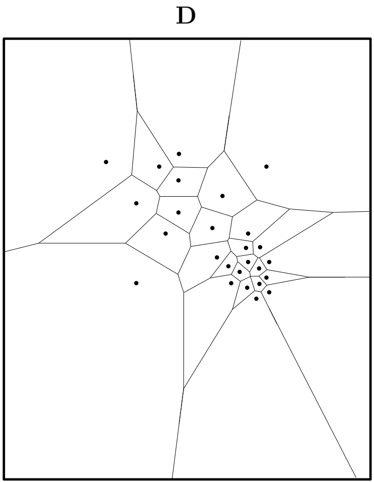
SVD Partitioning

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

SVD Partitioning

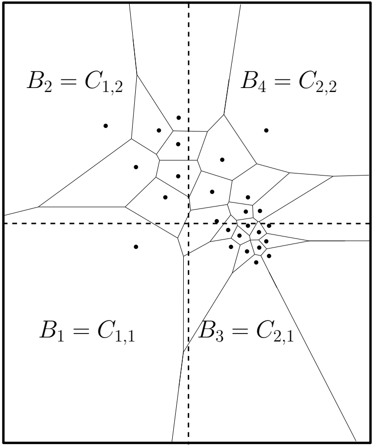
- Square Grid (SG)
- Minimum Space Grid (MinSG)
- Minimum Maximum Partition(MinMax)

Square Grid (SG)


D

Square Grid (SG)

G(D)



Square Grid (SG)

- Merits:

- Demerits:

Square Grid (SG)

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

Square Grid (SG)

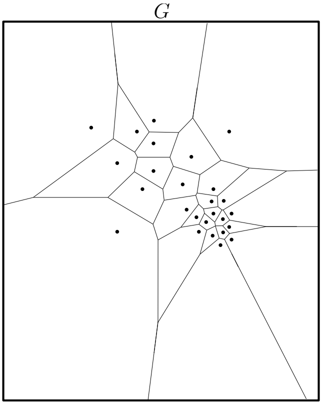
- Merits:
  - simple
  - minimum storage cost at client
- Demerits:
  - high storage and communication overheads, as well as expensive encryption cost because of highly unbalanced partitions when the data distribution is skewed

SVD Partitioning

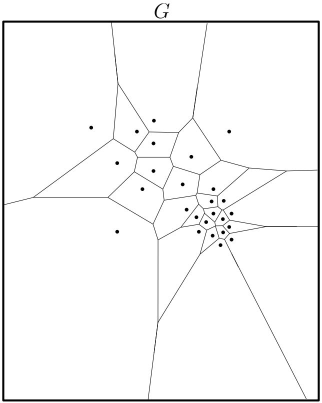
- Square Grid (SG)
- Minimum Space Grid (MinSG)
- Minimum Maximum Partition(MinMax)

Minimum Space Grid (MinSG)

Minimum Space Grid (MinSG)

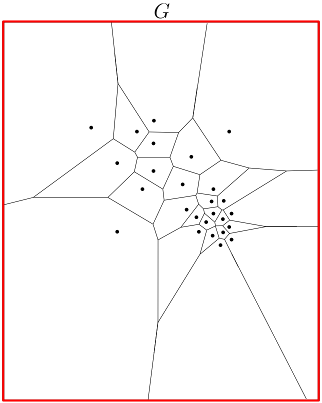


Minimum Space Grid (MinSG)



$|G| = 26$

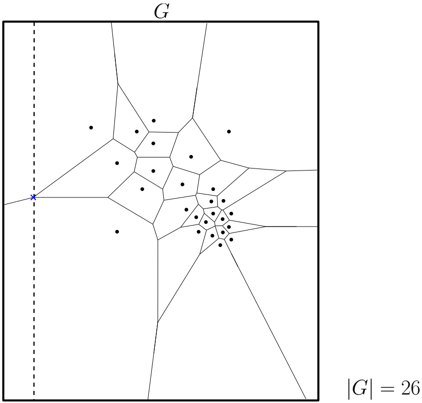
Minimum Space Grid (MinSG)



$|G| = 26$

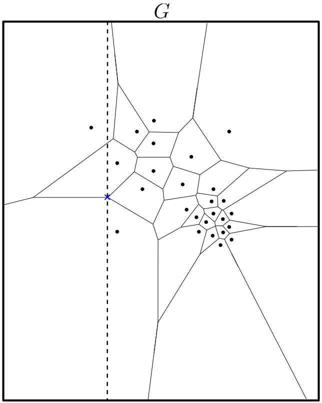
- A greedy algorithm: always split the maximum partition G_x into smaller partitions

Minimum Space Grid (MinSG)



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

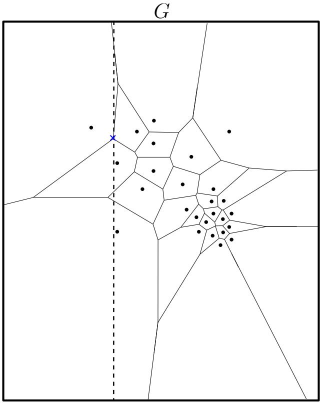
Minimum Space Grid (MinSG)



$|G| = 26$

- 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

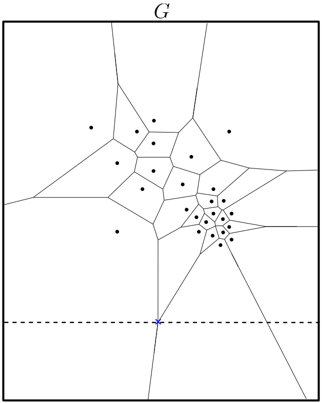
Minimum Space Grid (MinSG)



$|G| = 26$

- 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

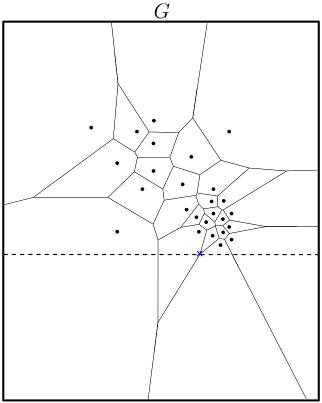
Minimum Space Grid (MinSG)



$|G| = 26$

- 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

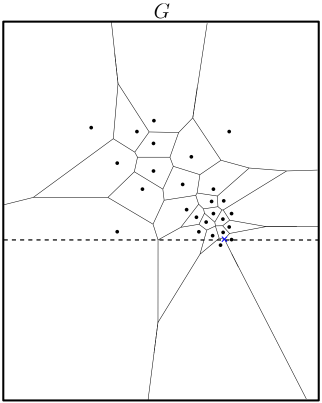
Minimum Space Grid (MinSG)



$|G| = 26$

- 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

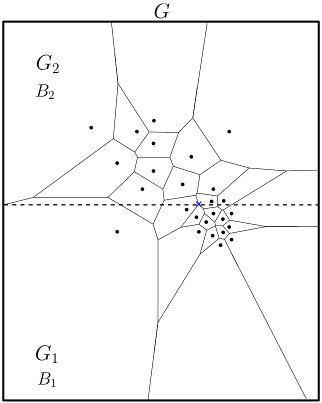
Minimum Space Grid (MinSG)



$|G| = 26$

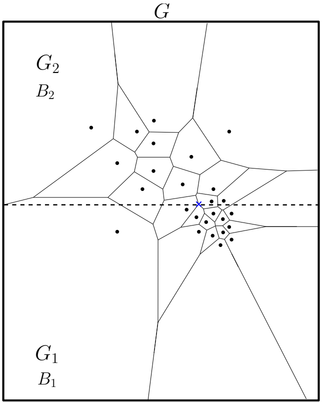
- A greedy algorithm: always split the maximum partition G_x into smaller partitions
- use a line going though the entire space and intersected with the voronoi vertex in B_x

Minimum Space Grid (MinSG)



- 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 ℓ that leads to the minimum maximum partition

Minimum Space Grid (MinSG)

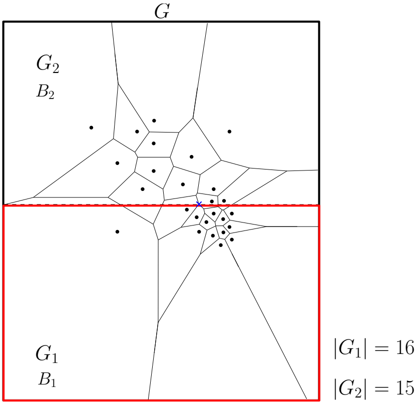


$|G_1| = 16$

$|G_2| = 15$

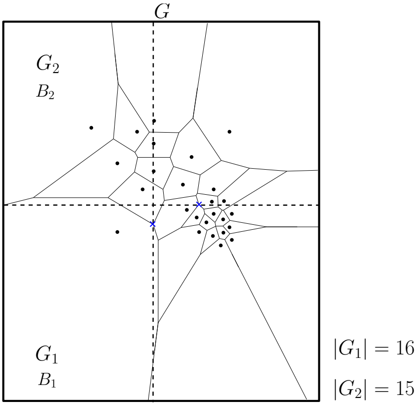
- A greedy algorithm: always split the maximum partition G_x into smaller partitions
- use a line going though the entire space and intersected with the voronoi vertex in B_x
- choose the ℓ that leads to the minimum maximum partition

Minimum Space Grid (MinSG)



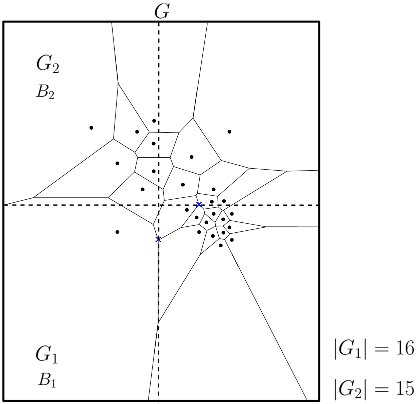
- A greedy algorithm: always split the maximum partition G_x into smaller partitions
- use a line going though the entire space and intersected with the voronoi vertex in B_x
- choose the ℓ that leads to the minimum maximum partition

Minimum Space Grid (MinSG)



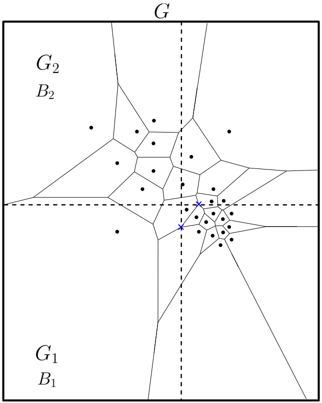
- A greedy algorithm: always split the maximum partition G_x into smaller partitions
- use a line going though the entire space and intersected with the voronoi vertex in B_x
- choose the ℓ that leads to the minimum maximum partition

Minimum Space Grid (MinSG)



- 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 ℓ that leads to the minimum maximum partition

Minimum Space Grid (MinSG)

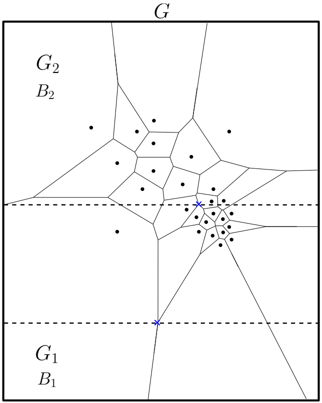


$|G_1| = 16$

$|G_2| = 15$

- 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 ℓ that leads to the minimum maximum partition

Minimum Space Grid (MinSG)

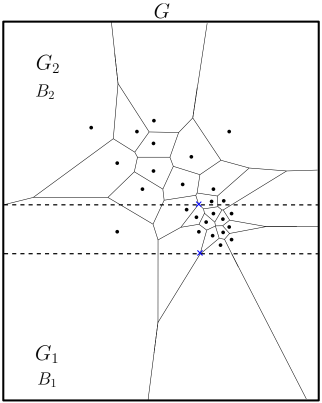


$|G_1| = 16$

$|G_2| = 15$

- 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 ℓ that leads to the minimum maximum partition

Minimum Space Grid (MinSG)

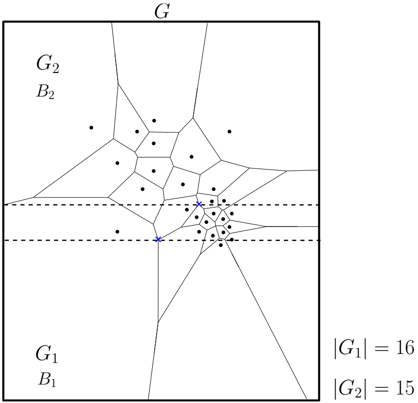


$|G_1| = 16$

$|G_2| = 15$

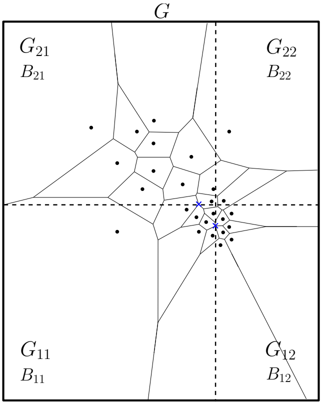
- A greedy algorithm: always split the maximum partition G_x into smaller partitions
- use a line going though the entire space and intersected with the voronoi vertex in B_x
- choose the ℓ that leads to the minimum maximum partition

Minimum Space Grid (MinSG)



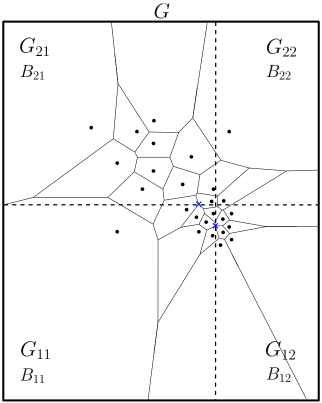
- A greedy algorithm: always split the maximum partition G_x into smaller partitions
- use a line going though the entire space and intersected with the voronoi vertex in B_x
- choose the ℓ that leads to the minimum maximum partition

Minimum Space Grid (MinSG)



- 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 ℓ that leads to the minimum maximum partition

Minimum Space Grid (MinSG)



$|G_{11}| = 11$
$|G_{12}| = 10$
$|G_{21}| = 14$
$|G_{22}| = 6$

- 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 ℓ that leads to the minimum maximum partition

Minimum Space Grid (MinSG)

- Merits:
- Demerits:

Minimum Space Grid (MinSG)

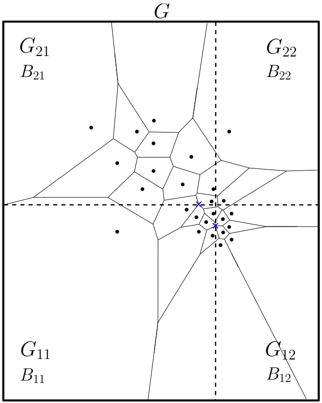
- Merits:
  - relatively balanced partitions: low storage and communication overheads, as well as cheap encryption cost
- Demerits:

Minimum Space Grid (MinSG)

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

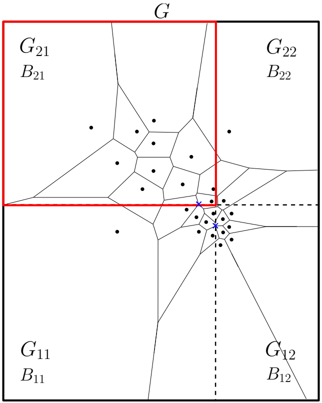
- Demerits:
  - complicated partitioning process
  - not most balanced: small-sized partitions introduced by some unnecessary splitting

Minimum Space Grid (MinSG)



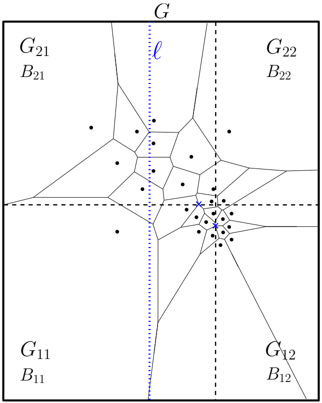
$|G_{11}| = 11$
$|G_{12}| = 10$
$|G_{21}| = 14$
$|G_{22}| = 6$

Minimum Space Grid (MinSG)

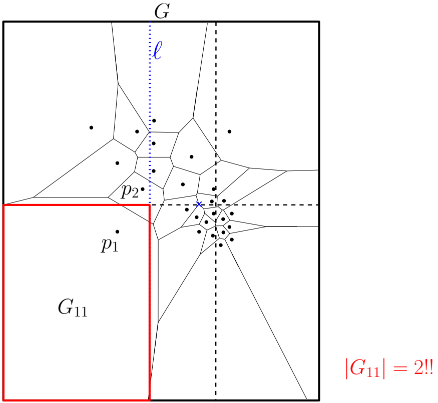


$|G_{11}| = 11$
$|G_{12}| = 10$
$|G_{21}| = 14$
$|G_{22}| = 6$

Minimum Space Grid (MinSG)



Minimum Space Grid (MinSG)



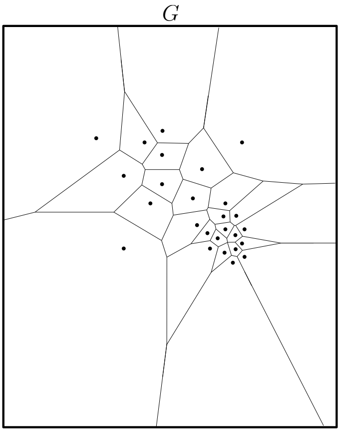
- We need a method that produce more balanced partitions!!

SVD Partitioning

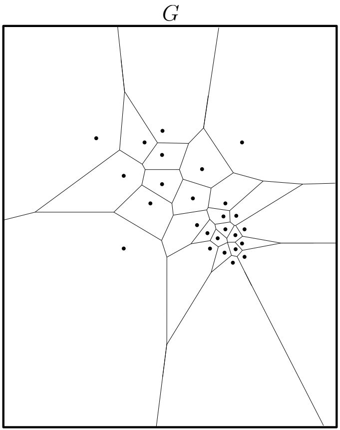
- Square Grid (SG)
- Minimum Space Grid (MinSG)
- Minimum Maximum Partition(MinMax)

Minimum Maximum Partition (MinMax)

Minimum Maximum Partition (MinMax)



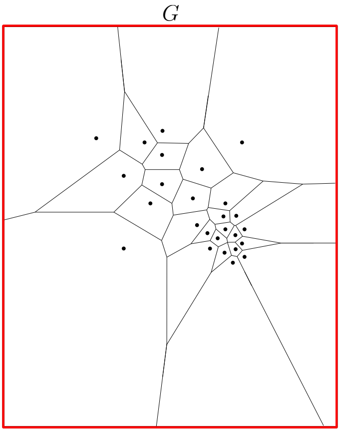
Minimum Maximum Partition (MinMax)



$|G| = 26$

- similar to MinSG in most part

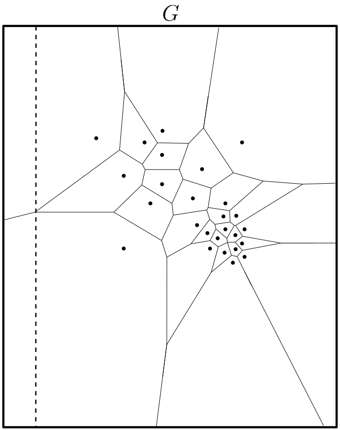
Minimum Maximum Partition (MinMax)



$|G| = 26$

- similar to MinSG in most part

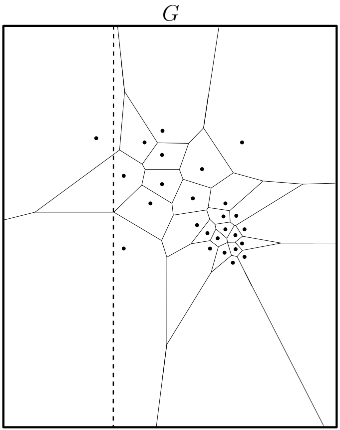
Minimum Maximum Partition (MinMax)



$|G| = 26$

- 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

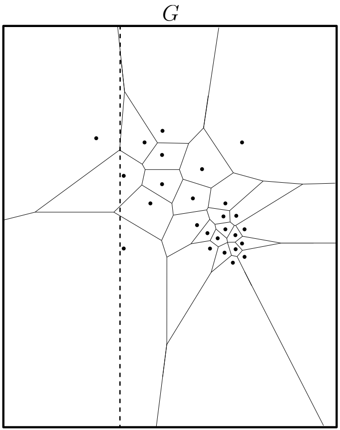
Minimum Maximum Partition (MinMax)



$|G| = 26$

- 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

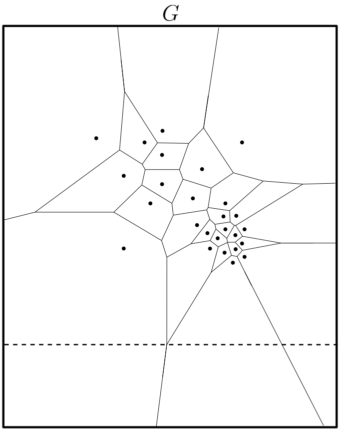
Minimum Maximum Partition (MinMax)



$|G| = 26$

- 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

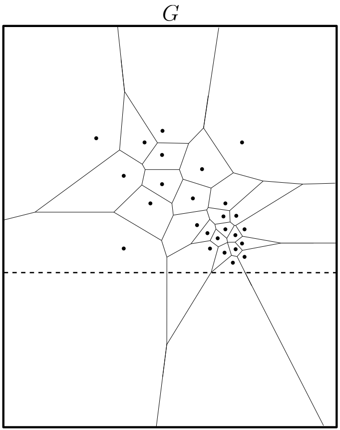
Minimum Maximum Partition (MinMax)



$|G| = 26$

- 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

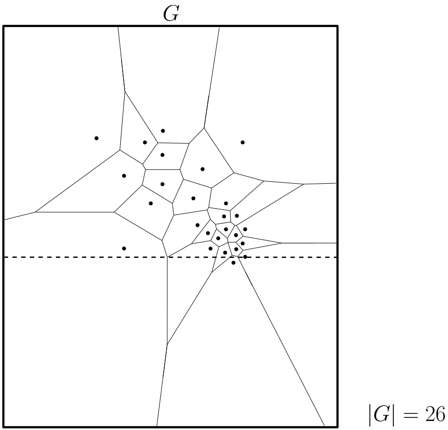
Minimum Maximum Partition (MinMax)



$|G| = 26$

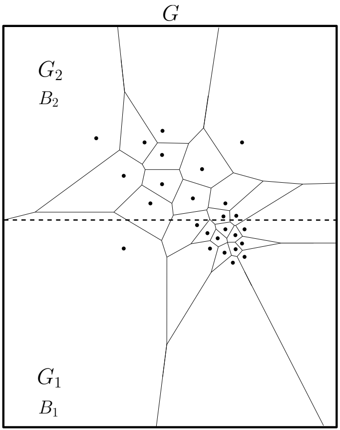
- similar to MinSG in most part
- use **segments** going though the space bounded by B_x instead of lines going though the entire space to split partitions

Minimum Maximum Partition (MinMax)



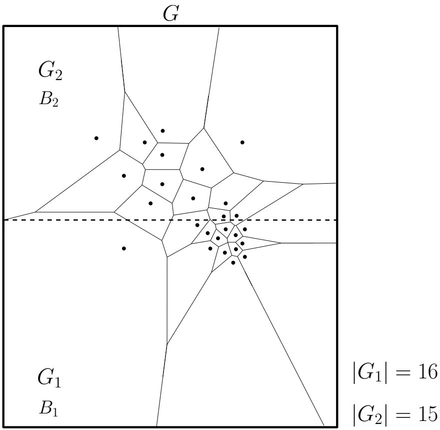
- similar to MinSG in most part
- use **segments** going though the space bounded by B_x instead of lines going though the entire space to split partitions

Minimum Maximum Partition (MinMax)



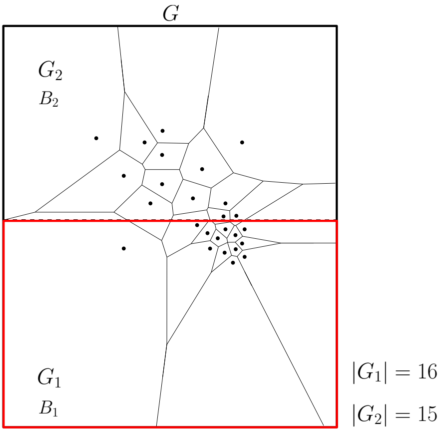
- similar to MinSG in most part
- use **segments** going though the space bounded by B_x instead of lines going though the entire space to split partitions

Minimum Maximum Partition (MinMax)



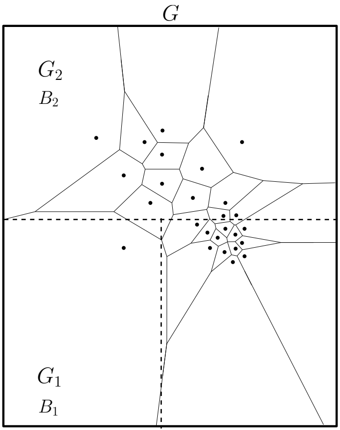
- similar to MinSG in most part
- use **segments** going though the space bounded by B_x instead of lines going though the entire space to split partitions

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)

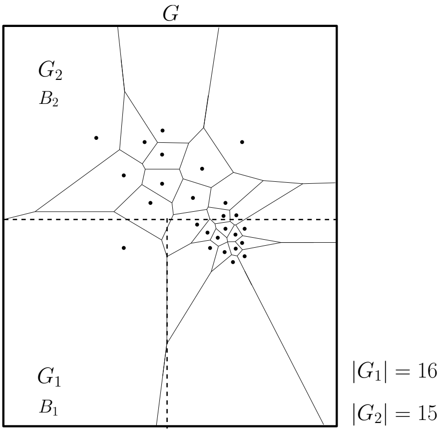


$|G_1| = 16$

$|G_2| = 15$

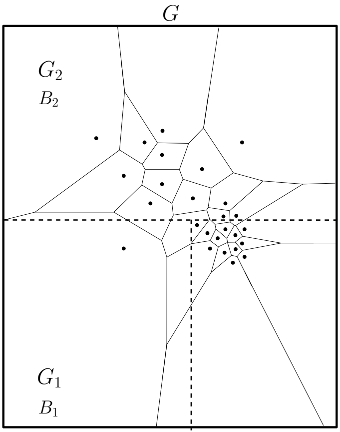
- similar to MinSG in most part
- use **segments** going though the space bounded by B_x instead of lines going though the entire space to split partitions

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)

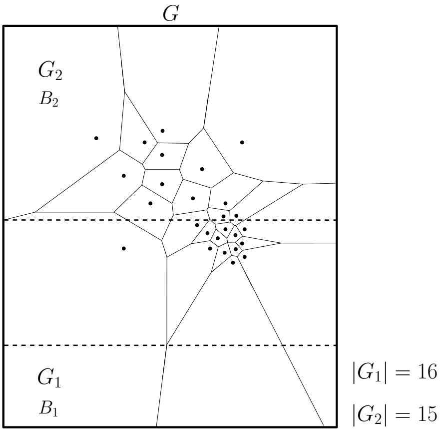


$|G_1| = 16$

$|G_2| = 15$

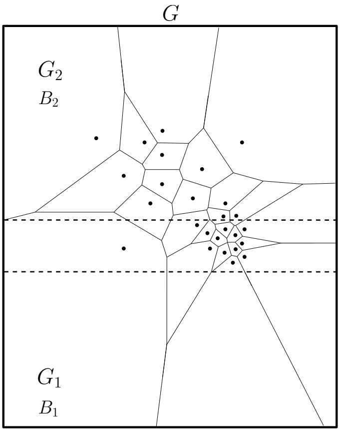
- similar to MinSG in most part
- use **segments** going though the space bounded by B_x instead of lines going though the entire space to split partitions

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)

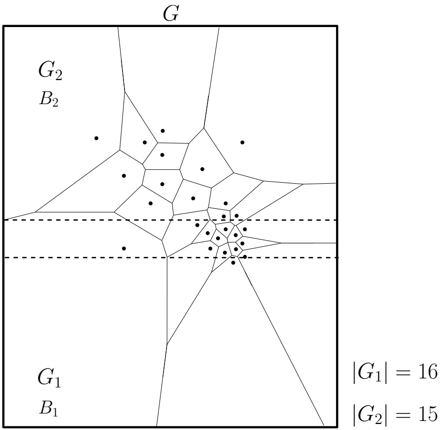


$|G_1| = 16$

$|G_2| = 15$

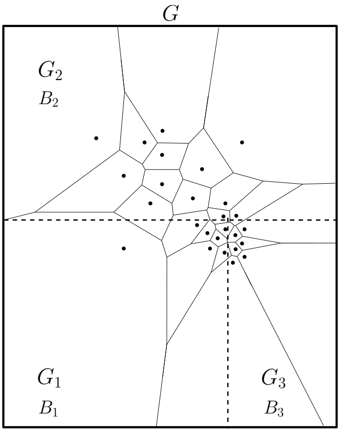
- similar to MinSG in most part
- use **segments** going though the space bounded by B_x instead of lines going though the entire space to split partitions

Minimum Maximum Partition (MinMax)



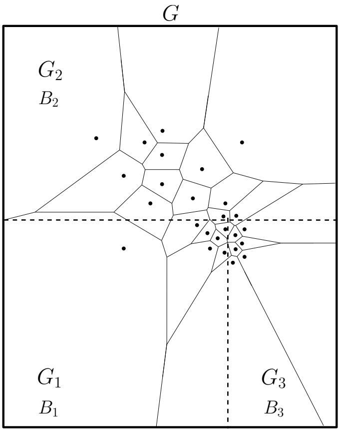
- similar to MinSG in most part
- use **segments** going though the space bounded by B_x instead of lines going though the entire space to split partitions

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

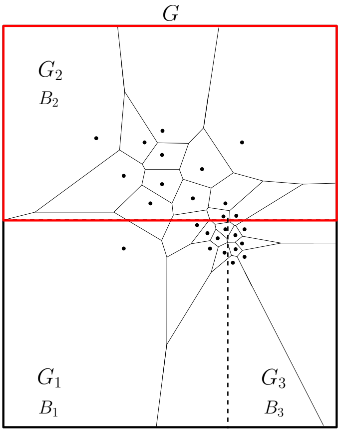
Minimum Maximum Partition (MinMax)



$|G_1| = 11$
$|G_2| = 15$
$|G_3| = 10$

- 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

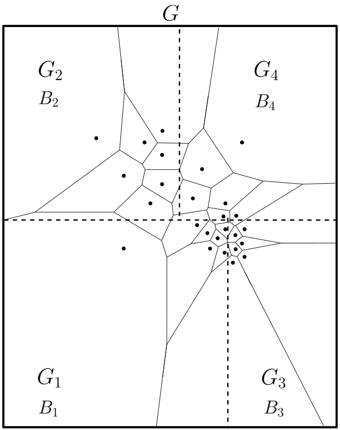
Minimum Maximum Partition (MinMax)



$|G_1| = 11$
$|G_2| = 15$
$|G_3| = 10$

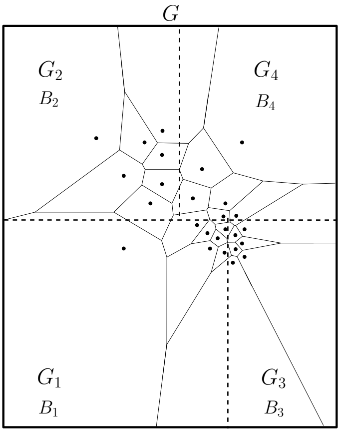
- similar to MinSG in most part
- use **segments** going though the space bounded by B_x instead of lines going though the entire space to split partitions

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)



$|G_1| = 11$
$|G_2| = 10$
$|G_3| = 10$
$|G_4| = 11$

- 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:
- Demerits:

Minimum Maximum Partition (MinMax)

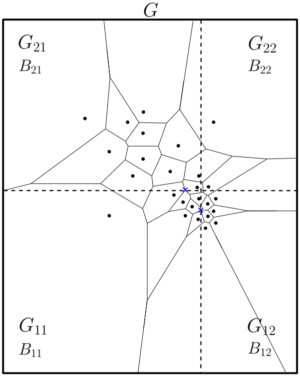
- Merits:
  - most balanced partitions: low storage and communication overheads, as well as cheap encryption cost
- Demerits:

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

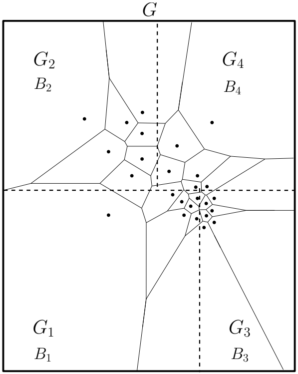
Comparison between MinSG and MinMax

Comparison between MinSG and MinMax



$|G_{11}| = 11$
$|G_{12}| = 10$
$|G_{21}| = 14$
$|G_{22}| = 6$

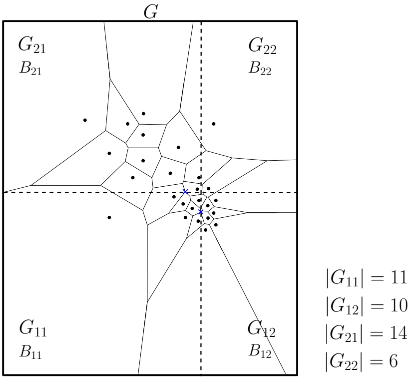
MinSG



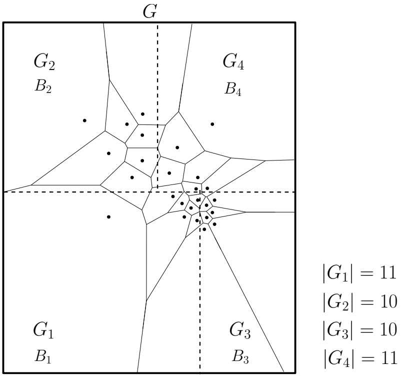
$|G_1| = 11$
$|G_2| = 10$
$|G_3| = 10$
$|G_4| = 11$

MinMax

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.

Experiment

Experiment

- We examine the three methods: SG, MinSG and MinMax.

Experiment

- 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.

Experiment

- 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

Experiment

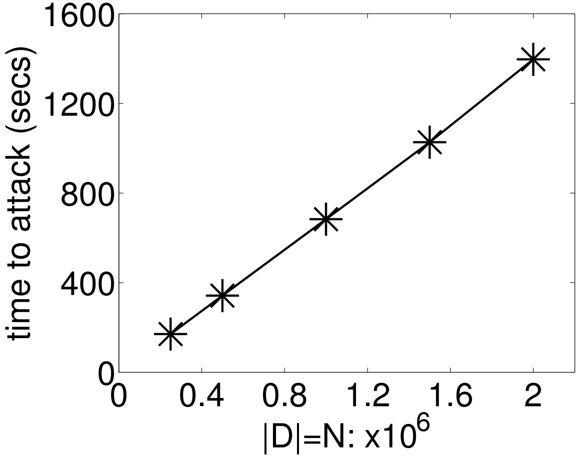
- We examine the three methods: SG, MinSG and MinMax.
- For each method, we test its running time of both partition phrase and encryption phrase, partition size, communication cost of both the preprocessing step and query step and query time.
- C++, Linux, Intel Xeon 3.07GHz CPU and 8GB memory
- Data 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_{max} and form smaller datasets based on D_{max}.

Experiment

- 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_{max} and form smaller datasets based on D_{max}.
- Default settings.

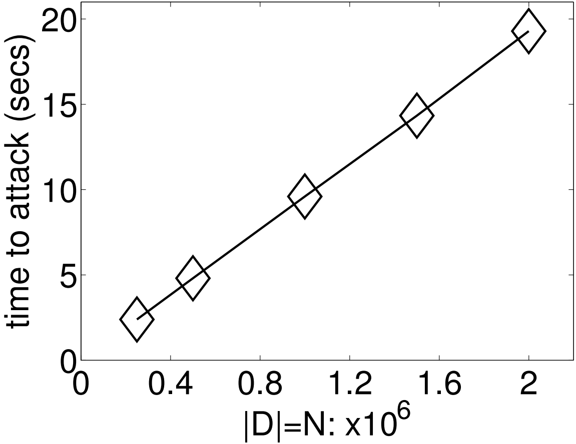
Symbol	Definition	Default Value
$\lvert D \rvert$	size of the dataset	10^6
k	number of partitions	625
DT	dataset type	CA

- Vary $|D|$: Wai Kit Wong, David Cheung, Ben Kao, Nikos Mamoulis: Secure kNN computation on encrypted databases. SIGMOD 2009



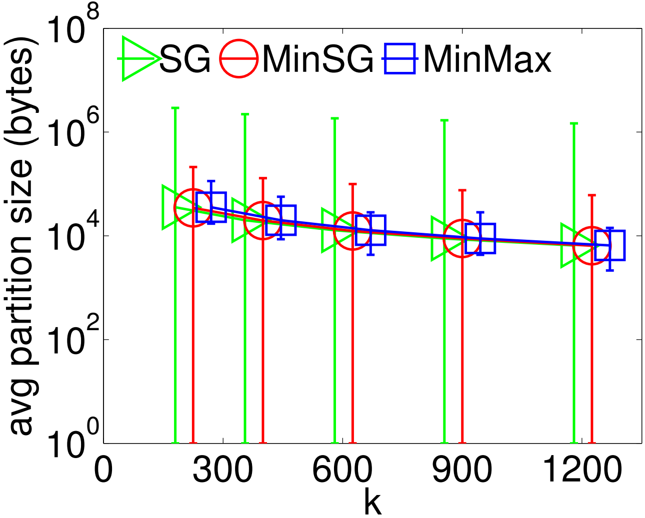
Attack on Existing SNN Methods

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



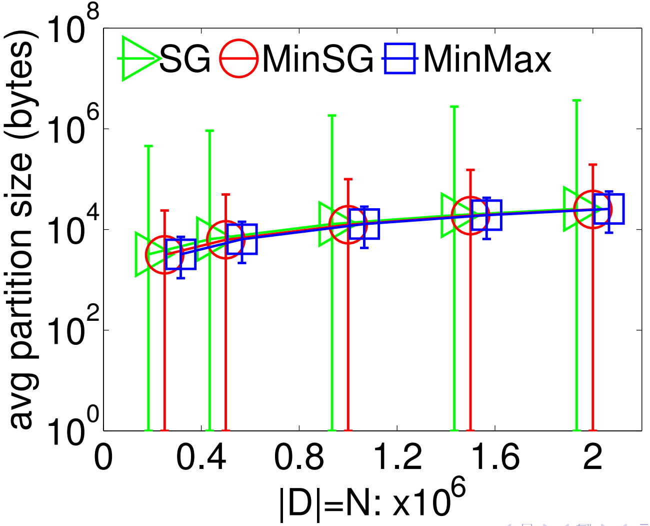
Partition size in different methods

- Vary *k*

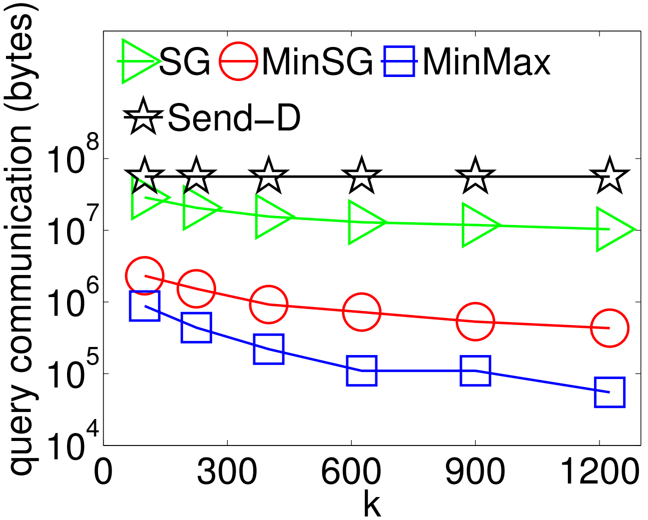


Partition size in different methods

- Vary $|D|$

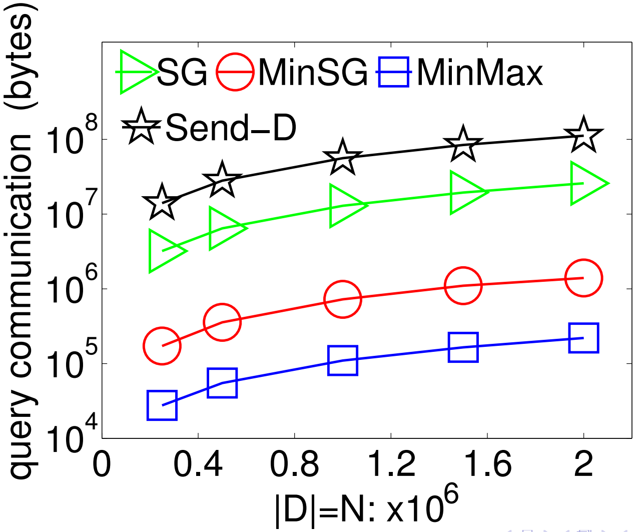


- Vary k


SG
MinSG
MinMax
Send–D
query communication (bytes)
10^8
10^7
10^6
10^5
10^4
0
300
600
900
1200
k

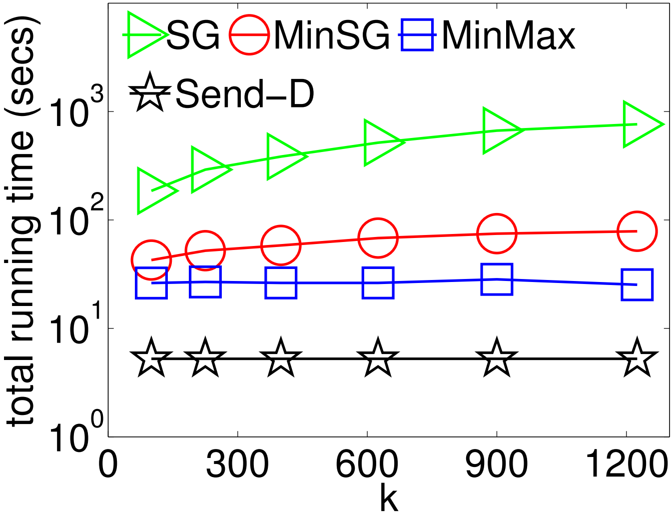
Query communication cost

- Vary $|D|$

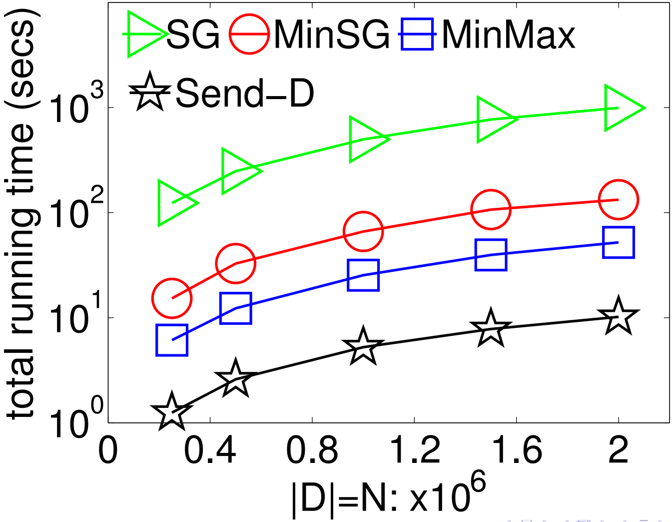


Total running time of the preprocessing step

- Vary *k*

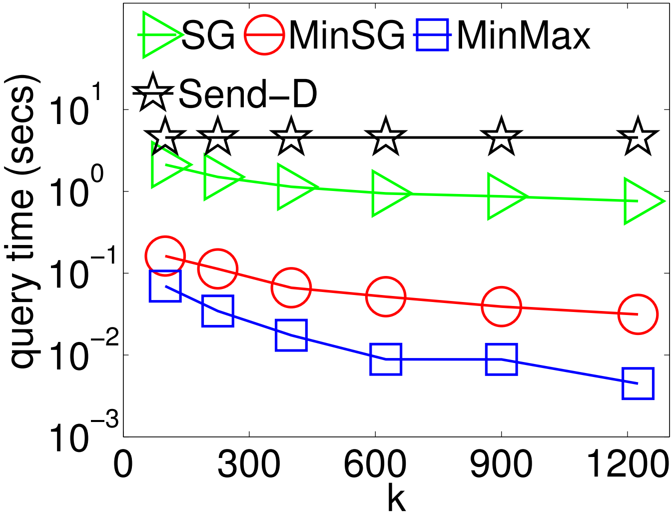


- Vary $|D|$


SG
MinSG
MinMax
Send–D
total running time (secs)
10^0
10^1
10^2
10^3
0
0.4
0.8
1.2
1.6
2
|D|=N: x10^6

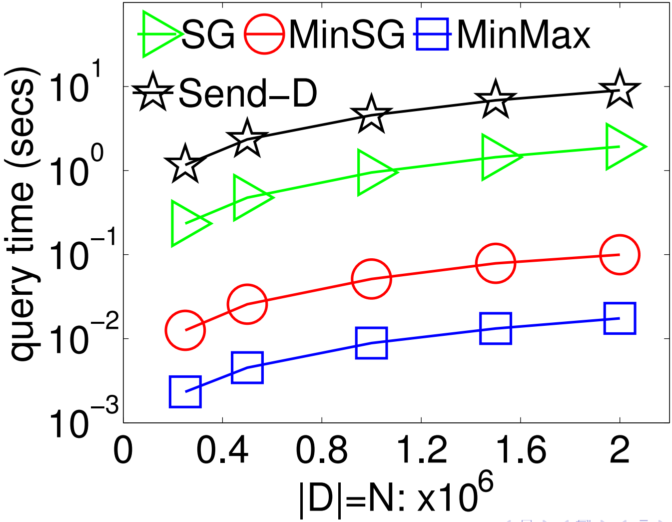
Query time for different methods

- Vary k

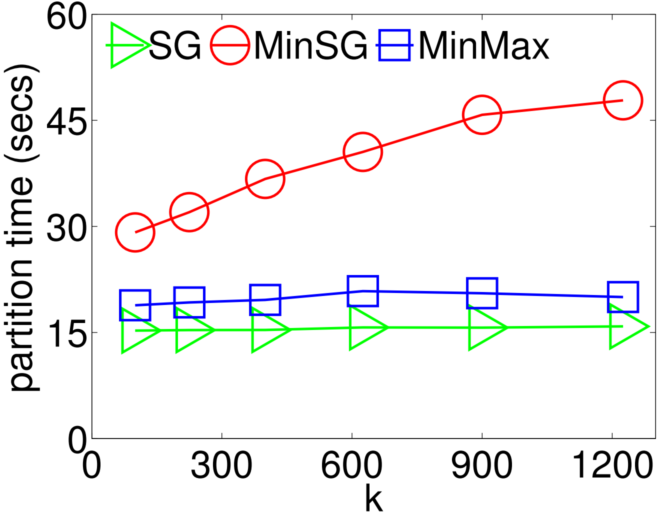


Query time for different methods

- Vary $|D|$

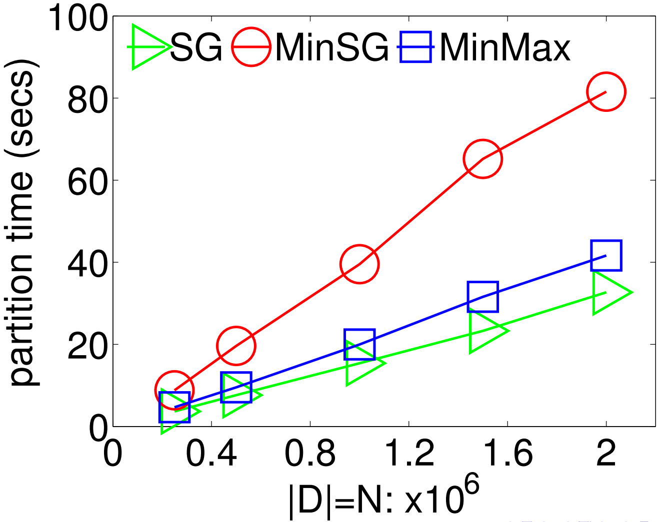


- Vary k


SG
MinSG
MinMax
60
45
30
15
0
partition time (secs)
0
300
600
900
1200
k

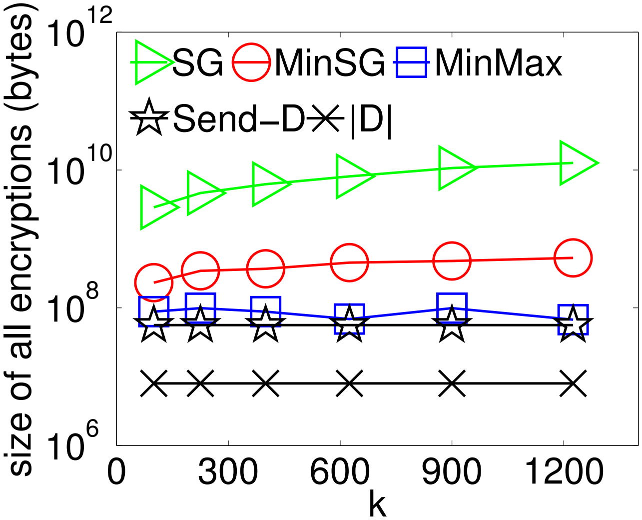
Running time of the partition phase

- Vary $|D|$



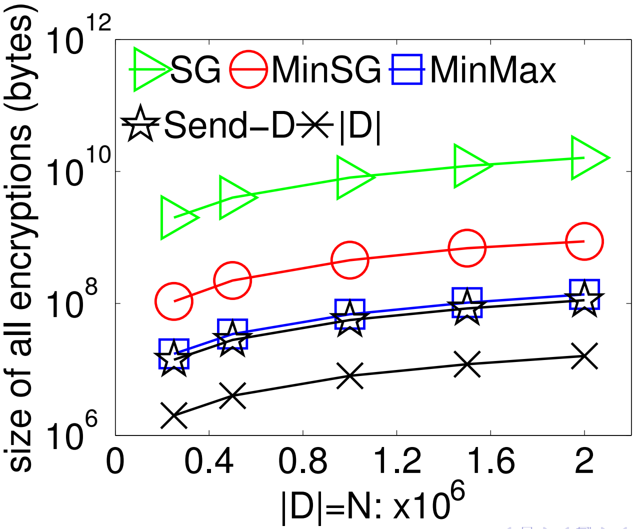
Total size of $E(D)$

- Vary k



Total size of $E(D)$

- 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?

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.

Conclusion

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

Conclusion

- 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