Dissertation Prospectus

Secure and Efficient Query Processing in Outsourced Databases

Range Queries [19, 21], Point Queries [21], kNN Queries, JOIN Queries

Dmytro Bogatov
dmytro@bu.edu

Built from 034dbe1e on January 4, 2022

Boston University
Graduate School of Arts and Sciences
Department of Computer Science
INTRODUCTION AND BACKGROUND
Motivation and overview

• With vast amounts of data, organizations choose to use cloud
• **Challenge**: solutions must be both **secure** and **efficient**
  
  • Query types: `SELECT * FROM t1`
  • Point queries: `WHERE zip = '02215'`
  • Range queries: `WHERE age BETWEEN 18 AND 65`
  • $k$NN queries: `ORDER BY location <-> '(29.9691,-95.6972)' LIMIT 5`
  • JOIN / GROUP BY queries: `INNER JOIN t2 ON (t1.k = t2.k) GROUP BY zip`

• Security models for an outsourced database system
  
  • **Snapshot** adversary: steal the hard drive and RAM snapshot
  • **Persistent** adversary: continuously monitor the entire server
Motivation and overview

- With vast amounts of data, organizations choose to use cloud
- **Challenge**: solutions must be both **secure** and **efficient**
- Query types: `SELECT * FROM t1`
  - Point queries: `WHERE zip = '02215'`
  - Range queries: `WHERE age BETWEEN 18 AND 65`
  - \(k\)NN queries: `ORDER BY location <-> '(29.9691,-95.6972)' LIMIT 5`
- JOIN / GROUP BY queries: `INNER JOIN t2 ON (t1.k = t2.k) GROUP BY zip`
- Security models for an outsourced database system
  - **Snapshot** adversary: steal the hard drive and RAM snapshot
  - **Persistent** adversary: continuously monitor the entire server
Motivation and overview

- With vast amounts of data, organizations choose to use cloud
- **Challenge:** solutions must be both **secure** and **efficient**
- Query types: `SELECT * FROM t1`
  - Point queries: `WHERE zip = '02215'`
  - Range queries: `WHERE age BETWEEN 18 AND 65`
  - $k$NN queries: `ORDER BY location <-> '(29.9691,-95.6972)' LIMIT 5`
  - JOIN / GROUP BY queries: `INNER JOIN t2 ON (t1.k = t2.k) GROUP BY zip`
- Security models for an outsourced database system
  - **Snapshot** adversary: steal the hard drive and RAM snapshot
  - **Persistent** adversary: continuously monitor the entire server
Motivation and overview

- With vast amounts of data, organizations choose to use cloud
- **Challenge:** solutions must be both *secure* and *efficient*
- Query types: `SELECT * FROM t1`
  - Point queries: `WHERE zip = '02215'`
  - Range queries: `WHERE age BETWEEN 18 AND 65`
  - \(k\)NN queries: `ORDER BY location <-> '(29.9691,-95.6972)' LIMIT 5`
  - JOIN / GROUP BY queries: `INNER JOIN t2 ON (t1.k = t2.k) GROUP BY zip`
- Security models for an outsourced database system
  - **Snapshot** adversary: steal the hard drive and RAM snapshot
  - **Persistent** adversary: continuously monitor the entire server
Motivation and overview

- With vast amounts of data, organizations choose to use cloud
- **Challenge:** solutions must be both *secure* and *efficient*
- Query types: `SELECT * FROM t1`
  - Point queries: `WHERE zip = '02215'`
  - Range queries: `WHERE age BETWEEN 18 AND 65`
  - kNN queries: `ORDER BY location <-> '(29.9691,-95.6972)' LIMIT 5`
  - JOIN / GROUP BY queries: `INNER JOIN t2 ON (t1.k = t2.k) GROUP BY zip`
- Security models for an outsourced database system
  - **Snapshot** adversary: steal the hard drive and RAM snapshot
  - **Persistent** adversary: continuously monitor the entire server
Motivation and overview

- With vast amounts of data, organizations choose to use cloud
- **Challenge**: solutions must be both **secure** and **efficient**
- Query types: `SELECT * FROM t1`
  - Point queries: `WHERE zip = '02215'`
  - Range queries: `WHERE age BETWEEN 18 AND 65`
  - $k$NN queries: `ORDER BY location <-> '(29.9691,-95.6972)' LIMIT 5`
  - JOIN / GROUP BY queries: `INNER JOIN t2 ON (t1.k = t2.k) GROUP BY zip`
- Security models for an outsourced database system
  - **Snapshot** adversary: steal the hard drive and RAM snapshot
  - Persistent adversary: continuously monitor the entire server
Motivation and overview

- With vast amounts of data, organizations choose to use cloud

  **Challenge:** solutions must be both **secure** and **efficient**

- Query types: `SELECT * FROM t1`
  - Point queries: `WHERE zip = '02215'`
  - Range queries: `WHERE age BETWEEN 18 AND 65`
  - $k$NN queries: `ORDER BY location <-> '(29.9691,-95.6972)' LIMIT 5`
  - JOIN / GROUP BY queries: `INNER JOIN t2 ON (t1.k = t2.k) GROUP BY zip`

- Security models for an outsourced database system
  - **Snapshot** adversary: steal the hard drive and RAM snapshot
  - **Persistent** adversary: continuously monitor the entire server
My work

Proposed thesis structure


Model: snapshot, query type: range


Model: persistent, query type: point and range

In-progress: Private $k$NN queries

Model: snapshot, query type: $k$NN

In-progress: Oblivious JOIN queries

Model: persistent, query type: JOIN
My work

Proposed thesis structure

*Proceedings of the VLDB Endowment, 12(8):933–947, 2019*

Model: snapshot, query type: range


Model: persistent, query type: point and range

In-progress: Private $k$NN queries

Model: snapshot, query type: $k$NN

In-progress: Oblivious JOIN queries

Model: persistent, query type: JOIN
Proposed thesis structure


Model: snapshot, query type: range


Model: persistent, query type: point and range

In-progress: Private $k$NN queries

Model: snapshot, query type: $k$NN

In-progress: Oblivious JOIN queries

Model: persistent, query type: JOIN
Proposed thesis structure


- **Model:** snapshot, **query type:** range


- **Model:** persistent, **query type:** point and range

In-progress: Private $k$NN queries

- **Model:** snapshot, **query type:** $k$NN

In-progress: Oblivious **JOIN** queries

- **Model:** persistent, **query type:** **JOIN**
My work

Proposed thesis structure


Model: snapshot, query type: range


Model: persistent, query type: point and range

In-progress: Private $k$NN queries
Model: snapshot, query type: $k$NN

In-progress: Oblivious JOIN queries
Model: persistent, query type: JOIN

A COMPARATIVE EVALUATION OF
ORDER-REVEALING ENCRYPTION
SCHEMES AND SECURE RANGE-QUERY
PROTOCOLS [19]
The problem

- Model: snapshot, query type: range
- Performance / security tradeoff
- Heterogeneous security definitions and leakage profiles
- **Performance not well-understood**
  - Some schemes are not even implemented
  - Prototype implementation at best
  - Not benchmarked against one another
  - Use different primitive implementations
  - Each claims to be practical and secure

Our solution

- Analyzed security and leakages of the constructions under a common framework
- Analyzed theoretically performance of the constructions
- Implemented and ran experiments
  - Implemented 5 OPE / ORE schemes and 5 range query protocols
  - Used same language, framework and primitive implementations
  - Benchmarked primitives execution times
  - Counted invocations of primitives and I/O requests
Survey of OPE/ORE schemes [19]

The problem

- Model: snapshot, query type: range
- Performance / security tradeoff
- Heterogeneous security definitions and leakage profiles
- Performance not well-understood
  - Some schemes are not even implemented
  - Prototype implementation at best
  - Not benchmarked against one another
  - Use different primitive implementations
  - Each claims to be practical and secure

Our solution

- Analyzed security and leakages of the constructions under a common framework
- Analyzed theoretically performance of the constructions
- Implemented and ran experiments
  - Implemented 5 OPE / ORE schemes and 5 range query protocols
  - Used same language, framework and primitive implementations
  - Benchmarked primitives execution times
  - Counted invocations of primitives and I/O requests
Epsolute: Efficiently Querying Databases While Providing Differential Privacy [21]
Motivation

The problem

- Previous solutions work in the snapshot model (adversary steals the hard drive)
- What about persistent adversary (malicious script with root permissions)?
  Model: persistent, query type: point and range
- Need to protect access pattern and communication volume
  - Using ORAM to hide the access pattern
    Expensive, each request costs $O(\log n)$
  - Adding fake records (noise) to the answer to hide the result size
    How much noise to add to have a guarantee and the least overhead?
    Adding a constant or a uniformly sampled noise is not an option
    Differential Privacy!
Motivation

The problem

• Previous solutions work in the snapshot model (adversary steals the hard drive)
• What about **persistent** adversary (malicious script with **root** permissions)?
  Model: persistent, query type: point and range
• Need to protect **access pattern** and **communication volume**
• Using ORAM to hide the access pattern
  Expensive, each request costs $O(\log n)$  
• Adding fake records (noise) to the answer to hide the result size
  How much noise to add to have a guarantee and the least overhead?
  Adding a constant or a uniformly sampled noise is not an option
  Differential Privacy!
The problem

• Previous solutions work in the snapshot model (adversary steals the hard drive)
• What about persistent adversary (malicious script with root permissions)?
  Model: persistent, query type: point and range
• Need to protect access pattern and communication volume
• Using ORAM to hide the access pattern
  Expensive, each request costs $O(\log n)$
• Adding fake records (noise) to the answer to hide the result size
  How much noise to add to have a guarantee and the least overhead?
  Adding a constant or a uniformly sampled noise is not an option
  Differential Privacy!
Definition (Differential Privacy, adapted from [5, 6])

A randomized algorithm $A$ is $(\epsilon, \delta)$-differentially private if for all $D_1 \sim D_2 \in \mathcal{X}^n$, and for all subsets $O$ of the output space of $A$, 

$$\Pr[A(D_1) \in O] \leq \exp(\epsilon) \cdot \Pr[A(D_2) \in O] + \delta.$$ 

How to make sense of it?

- Differential Privacy is a property of an algorithm. What about $\epsilon$ and $\delta$?
- How to construct such an algorithm? Laplace Perturbation Method!
- What if negative value is sampled? Cannot truncate one side, must shift entire distribution.
Definition (Differential Privacy, adapted from [5, 6])

A randomized algorithm $A$ is $(\epsilon, \delta)$-differentially private if for all $D_1 \sim D_2 \in \mathcal{X}^n$, and for all subsets $O$ of the output space of $A$,

$$\Pr [A(D_1) \in O] \leq \exp(\epsilon) \cdot \Pr [A(D_2) \in O] + \delta.$$  

How to make sense of it?

• Differential Privacy is a property of an algorithm

  What about $\epsilon$ and $\delta$?

• How to construct such an algorithm?

  Laplace Perturbation Method!

• What if negative value is sampled?

  Cannot truncate one side, must shift entire distribution
Definition \((\textbf{Differential Privacy}, \ \textit{adapted from [5, 6]})\)

A randomized algorithm \(A\) is \((\epsilon, \delta)\)-differentially private if for all \(D_1 \sim D_2 \in \mathcal{X}^n\), and for all subsets \(O\) of the output space of \(A\),

\[
\Pr [A(D_1) \in O] \leq \exp(\epsilon) \cdot \Pr [A(D_2) \in O] + \delta.
\]

How to make sense of it?

\begin{itemize}
\item Differential Privacy is a property of an algorithm
  What about \(\epsilon\) and \(\delta\)?
\item How to construct such an algorithm?
  Laplace Perturbation Method!
\item What if negative value is sampled?
  Cannot truncate one side, must shift entire distribution
\end{itemize}
Definition (**Differential Privacy**, adapted from [5, 6])

A randomized algorithm $A$ is $(\epsilon, \delta)$-differentially private if for all $D_1 \sim D_2 \in \mathcal{X}^n$, and for all subsets $\mathcal{O}$ of the output space of $A$,

$$\Pr[A(D_1) \in \mathcal{O}] \leq \exp(\epsilon) \cdot \Pr[A(D_2) \in \mathcal{O}] + \delta.$$

**How to make sense of it?**

- Differential Privacy is a property of an algorithm
  - What about $\epsilon$ and $\delta$?

- How to construct such an algorithm?
  - Laplace Perturbation Method!

- What if negative value is sampled?
  - Cannot truncate one side, must shift entire distribution
Definition (**Differential Privacy**, adapted from [5, 6])

A randomized algorithm \( A \) is \((\epsilon, \delta)\)-differentially private if for all \( D_1 \sim D_2 \in \mathcal{X}^n \), and for all subsets \( O \) of the output space of \( A \),

\[
\Pr [A(D_1) \in O] \leq \exp(\epsilon) \cdot \Pr [A(D_2) \in O] + \delta .
\]

How to make sense of it?

- **Differential Privacy** is a property of an algorithm
  
  What about \( \epsilon \) and \( \delta \)?
- How to construct such an algorithm?
  
  Laplace Perturbation Method!
- What if negative value is sampled?
  
  Cannot truncate one side, must shift entire distribution
Definition (Computationally Differentially Private Outsourced Database System (CDP-ODB))

We say that an outsourced database system $\Pi$ is $(\epsilon, \delta)$-computationally differentially private (a.k.a. CDP-ODB) if for every polynomial time distinguishing adversary $A$, for every neighboring databases $D \sim D'$, and for every query sequence $q_1, \ldots, q_m \in Q^m$ where $m = \text{poly}(\lambda)$,

$$\Pr \left[ A(1^\lambda, \text{VIEW}_\Pi, S(\mathcal{D}, q_1, \ldots, q_m)) = 1 \right] \leq \exp \epsilon \cdot \Pr \left[ A(1^\lambda, \text{VIEW}_\Pi, S(\mathcal{D}', q_1, \ldots, q_m)) = 1 \right] + \delta + \text{negl}(\lambda),$$

the probability is over the randomness of the distinguishing adversary $A$ and the protocol $\Pi$.

Note:

- Entire view of the adversary is DP-protected
- Implies protection against communication volume and access pattern leakages
- Query sequence $q_1, \ldots, q_m \in Q^m$ is fixed (more on that next)
- $\text{negl}(\lambda)$ needed for the computational (as opposed to information-theoretical) DP definition
Definition (Computationally Differentially Private Outsourced Database System (CDP-ODB))

We say that an outsourced database system $\Pi$ is $(\epsilon, \delta)$-computationally differentially private (a.k.a. CDP-ODB) if for every polynomial time distinguishing adversary $A$, for every neighboring databases $D \sim D'$, and for every query sequence $q_1, \ldots, q_m \in Q^m$ where $m = \text{poly}(\lambda)$,

$$
\Pr[A(1^\lambda, \text{VIEW}_{\Pi,S}(D, q_1, \ldots, q_m)) = 1] \leq \exp \epsilon \cdot \Pr[A(1^\lambda, \text{VIEW}_{\Pi,S}(D', q_1, \ldots, q_m)) = 1] + \delta + \text{negl}(\lambda),
$$

the probability is over the randomness of the distinguishing adversary $A$ and the protocol $\Pi$.

Note:

- Entire view of the adversary is DP-protected
- Implies protection against communication volume and access pattern leakages
- Query sequence $q_1, \ldots, q_m \in Q^m$ is fixed (more on that next)
- $\text{negl}(\lambda)$ needed for the computational (as opposed to information-theoretical) DP definition
On impossibility of adaptive queries

Why is the query sequence $q_1, \ldots, q_m \in Q^m$ fixed?

- Suppose neighboring medical databases differ in one record with a rare diagnosis “Alzheimer’s disease”
- A medical professional, who is a user, not an adversary queries the database
  - for that diagnosis first
    SELECT name FROM patients WHERE condition = 'ALZ'
  - if there is a record, she queries the senior patients next
    SELECT name FROM patients WHERE age >= 65
  - otherwise she queries the general population, resulting in many more records
    SELECT name FROM patients
- Adversary can know the answer to the first query by observing result size of the second
- Efficient system cannot return nearly the same number of records in both cases, thus, the adversary can distinguish
Why is the query sequence $q_1, \ldots, q_m \in Q^m$ fixed?

- Suppose neighboring medical databases differ in one record with a rare diagnosis “Alzheimer’s disease”
- A medical professional, who is a user, not an adversary queries the database
  - for that diagnosis first
    \[
    \text{SELECT name FROM patients WHERE condition = 'ALZ'}
    \]
  - if there is a record, she queries the senior patients next
    \[
    \text{SELECT name FROM patients WHERE age >= 65}
    \]
  - otherwise she queries the general population, resulting in many more records
    \[
    \text{SELECT name FROM patients}
    \]
- Adversary can know the answer to the first query by observing result size of the second
- Efficient system cannot return nearly the same number of records in both cases, thus, the adversary can distinguish
On impossibility of adaptive queries

Why is the query sequence $q_1, \ldots, q_m \in Q^m$ fixed?

- Suppose neighboring medical databases differ in one record with a rare diagnosis “Alzheimer’s disease”
- A medical professional, who is a user, not an adversary queries the database
  - for that diagnosis first
    
    ```sql
    SELECT name FROM patients WHERE condition = 'ALZ'
    ```
  - if there is a record, she queries the senior patients next
    
    ```sql
    SELECT name FROM patients WHERE age >= 65
    ```
  - otherwise she queries the general population, resulting in many more records
    
    ```sql
    SELECT name FROM patients
    ```
- Adversary can know the answer to the first query by observing result size of the second
- Efficient system cannot return nearly the same number of records in both cases, thus, the adversary can distinguish
Single-Threaded $\epsilon$psolute protocol

Query: “Salaries $40K–$50K"
Parallel $\varepsilon$psolute

- Single-threaded version is prohibitively slow, must parallelize
  Assume single-threaded solution generates $r = 1500$ real and $f = 500$ noisy records
- Split $U$ and $S$ state into $m$ ORAMs, run as separate machines (assume $m = 4$)
- Partition records randomly (by ID) into $m$ partitions, generate $m$ inverted indexes
- What to do about $DS$?

**No-$\gamma$ method:** $DS$ per ORAM
- Composition of disjoint datasets: take max $\epsilon$
- Each ORAM incurs noise comparable to $f$
- Win by splitting ORAM work $r$ into $m$ partitions and lose by multiplying noise $f$ times $m$
- That is, each ORAM is processing $\frac{r}{m} + f = 875$ records in parallel

**$\gamma$-method:** shared $DS$
- Same number of total records per ORAM
- Generated noise is larger than $f$ (say, $2f$)
- But it is split among $m$ ORAMs
- That is, each ORAM is processing $\frac{r+2f}{m} = 625$ records in parallel
Parallel $\varepsilon$psolute

- Single-threaded version is prohibitively slow, must parallelize
  Assume single-threaded solution generates $r = 1500$ real and $f = 500$ noisy records
- Split $U$ and $S$ state into $m$ ORAMs, run as separate machines (assume $m = 4$)
- Partition records randomly (by ID) into $m$ partitions, generate $m$ inverted indexes
- What to do about $DS$?

No-$\gamma$ method: $DS$ per ORAM
- Composition of disjoint datasets: take max $\epsilon$
- Each ORAM incurs noise comparable to $f$
- Win by splitting ORAM work $r$ into $m$ partitions and lose by multiplying noise $f$ times $m$
- That is, each ORAM is processing $\frac{r}{m} + f = 875$ records in parallel

$\gamma$-method: shared $DS$
- Same number of total records per ORAM
- Generated noise is larger than $f$ (say, $2f$)
- But it is split among $m$ ORAMs
- That is, each ORAM is processing $\frac{r+2f}{m} = 625$ records in parallel
Parallel $\varepsilon$psolute

- Single-threaded version is prohibitively slow, must parallelize
  Assume single-threaded solution generates $r = 1500$ real and $f = 500$ noisy records
- Split $\mathcal{U}$ and $\mathcal{S}$ state into $m$ ORAMs, run as separate machines (assume $m = 4$)
- Partition records randomly (by ID) into $m$ partitions, generate $m$ inverted indexes
- What to do about $\mathcal{D}$S?

**No-$\gamma$ method: $\mathcal{D}$S per ORAM**
- Composition of disjoint datasets: take max $\epsilon$
- Each ORAM incurs noise comparable to $f$
- Win by splitting ORAM work $r$ into $m$ partitions and lose by multiplying noise $f$ times $m$
- That is, each ORAM is processing $\frac{r}{m} + f = 875$ records in parallel

**$\gamma$-method: shared $\mathcal{D}$S**
- Same number of total records per ORAM
- Generated noise is larger than $f$ (say, $2f$)
- But it is split among $m$ ORAMs
- That is, each ORAM is processing $\frac{r+2f}{m} = 625$ records in parallel
Parallel $\varepsilon$psolute diagram (with improvements)

1 Query:

```
ages 18 to 21
```

Untrusted server party $S$

Trusted user party $U$

User

Client

Application

Lightweight ORAM machine

Lightweight ORAM machine

B+ tree

ORAM requests:

```
ORAM IDs
Block IDs
```

5 ORAM GET requests

Computing the amount of noise

6 pruning fake records

5 ORAM GET requests

KVS Store

DP histogram

DP tree

6 pruning fake records

KVS Store

KVS Store

KVS Store
Experiments: against other mechanisms

Different range-query mechanisms (log scale). Default setting: $10^6$ 4 KiB uniformly-sampled records with the range $10^4$. 

- MySQL: 97 ms
- PostgreSQL: 220 ms
- Epsolute: 840 ms
- Linear Scan: 15 s
- Shrinkwrap*: 19.5 min
Experiments: scalability

Scalability measurements for $\Pi_\gamma$ (shared $\mathcal{DS}$) and $\Pi_{\text{no-}\gamma}$ ($\mathcal{DS}$ per ORAM)
WORK-IN-PROGRESS:
PRIVATE $k$NN QUERIES
• Model: **snapshot**, query type: **$k$NN** in arbitrary dimensions
  
  • Input: vector of real numbers, query: return $k$ “closest” inputs to given vector
  
  Distance can be $L_p$ (usually, Euclidean, $p = 2$) or inner (dot) product
  
  • Applications range from similarity search to geographical search
  
  Document is a vector of words/features/topics, query is to find $k$ most similar documents
  
  Object on a map is a 2D vector, query is to find $k$ nearest locations
  
  • **Approximate distance-comparison preserving encryption (DCPE) scheme** on input and queries
    
    $\forall x, y, z \in X : \text{DIST}(x, y) < \text{DIST}(x, z) - \beta \implies \text{DIST}(f(x), f(y)) < \text{DIST}(f(x), f(z))$
    
  • Prove theoretically and observe empirically how accuracy of search and efficiency of attacks drop with higher security

DCPE | TREC and FAISS | Intermediate results plot
General idea

- Model: snapshot, query type: $k$NN in arbitrary dimensions
- Input: vector of real numbers, query: return $k$ “closest” inputs to given vector
  Distance can be $L_p$ (usually, Euclidean, $p = 2$) or inner (dot) product
- Applications range from similarity search to geographical search
  Document is a vector of words/features/topics, query is to find $k$ most similar documents
  Object on a map is a 2D vector, query is to find $k$ nearest locations
- Approximate distance-comparison preserving encryption (DCPE) scheme on input and queries
  $$\forall x, y, z \in X : \text{DIST}(x, y) < \text{DIST}(x, z) - \beta \implies \text{DIST}(f(x), f(y)) < \text{DIST}(f(x), f(z))$$
- Prove theoretically and observe empirically how accuracy of search and efficiency of attacks drop with higher security
General idea

- Model: **snapshot**, query type: \( k \text{NN} \) in arbitrary dimensions
- Input: vector of real numbers, query: return \( k \) “closest” inputs to given vector
  Distance can be \( L_p \) (usually, Euclidean, \( p = 2 \)) or inner (dot) product
- Applications range from similarity search to geographical search
  Document is a vector of words/features/topics, query is to find \( k \) most similar documents
  Object on a map is a 2D vector, query is to find \( k \) nearest locations
- **Approximate distance-comparison preserving encryption (DCPE) scheme** on input and queries

\[ \forall x, y, z \in X : \text{DIST}(x, y) < \text{DIST}(x, z) - \beta \implies \text{DIST}(f(x), f(y)) < \text{DIST}(f(x), f(z)) \]

- Prove theoretically and observe empirically how accuracy of search and efficiency of attacks drop with higher security

- DCPE
- TREC and FAISS
- Intermediate results plot
General idea

- Model: snapshot, query type: $k$NN in arbitrary dimensions
- Input: vector of real numbers, query: return $k$ “closest” inputs to given vector
  Distance can be $L_p$ (usually, Euclidean, $p = 2$) or inner (dot) product
- Applications range from similarity search to geographical search
  Document is a vector of words/features/topics, query is to find $k$ most similar documents
  Object on a map is a 2D vector, query is to find $k$ nearest locations
- Approximate distance-comparison preserving encryption (DCPE) scheme on input and queries
  \[
  \forall x, y, z \in X : \text{DIST}(x, y) < \text{DIST}(x, z) - \beta \implies \text{DIST}(f(x), f(y)) < \text{DIST}(f(x), f(z))
  \]
- Prove theoretically and observe empirically how accuracy of search and efficiency of attacks drop with higher security
General idea

- Model: snapshot, query type: \( k \text{NN} \) in arbitrary dimensions
- Input: vector of real numbers, query: return \( k \) “closest” inputs to given vector
  Distance can be \( L_p \) (usually, Euclidean, \( p = 2 \)) or inner (dot) product
- Applications range from similarity search to geographical search
  Document is a vector of words/features/topics, query is to find \( k \) most similar documents
  Object on a map is a 2D vector, query is to find \( k \) nearest locations
- \textit{Approximate distance-comparison preserving encryption (DCPE) scheme} on input and queries
  \[
  \forall x, y, z \in \mathbb{X} : \text{DIST} (x, y) < \text{DIST} (x, z) - \beta \implies \text{DIST} (f(x), f(y)) < \text{DIST} (f(x), f(z))
  \]
- Prove theoretically and observe empirically how accuracy of search and efficiency of attacks drop with higher security
WORK-IN-PROGRESS:
oblivious JOINs
General idea

• Model: **persistent**, query type: **inner equi-JOIN**

• Input: two tables $T_1$ and $T_2$, query: return a cross-product of $T_1$ and $T_2$ where $T_1.k = T_2.k$
  We may also consider **SELECT JOIN: WHERE $T_1.k = T_2.k$ AND $T_1.a = 10$**

• **Challenge**: produce JOIN result hiding both access pattern and result size

• **Proposed solution**:
  • use enclave (SGX) and oblivious primitives (sort, compaction)
  • construct index over join keys, add DP noise to it
  • partition the data by keys to fit a partition in the enclave
  • consolidate sparse keys as an optimization
  • do inner join within partition

⇒ Detailed Algorithm
General idea

• Model: persistent, query type: inner equi-JOIN

• Input: two tables $T_1$ and $T_2$, query: return a cross-product of $T_1$ and $T_2$ where $T_1.k = T_2.k$
  We may also consider \texttt{SELECT JOIN WHERE T1.k = T2.k AND T1.a = 10}

• Challenge: produce JOIN result hiding both access pattern and result size

• Proposed solution:
  • use enclave (SGX) and oblivious primitives (sort, compaction)
  • construct index over join keys, add DP noise to it
  • partition the data by keys to fit a partition in the enclave
  • consolidate sparse keys as an optimization
  • do inner join within partition

Detailed Algorithm
General idea

- Model: **persistent**, query type: **inner equi-JOIN**
- Input: two tables $T_1$ and $T_2$, query: return a cross-product of $T_1$ and $T_2$ where $T_1.k = T_2.k$
  We may also consider `SELECT JOIN WHERE T_1.k = T_2.k AND T_1.a = 10`
- **Challenge**: produce JOIN result hiding both access pattern and result size
- **Proposed solution**:
  - use enclave (SGX) and oblivious primitives (sort, compaction)
  - construct index over join keys, add DP noise to it
  - partition the data by keys to fit a partition in the enclave
  - consolidate sparse keys as an optimization
  - do inner join within partition
General idea

- **Model:** persistent, query type: inner equi-JOIN
- **Input:** two tables $T_1$ and $T_2$, query: return a cross-product of $T_1$ and $T_2$ where $T_1.k = T_2.k$
- **Challenge:** produce JOIN result hiding both access pattern and result size
- **Proposed solution:**
  - use enclave (SGX) and oblivious primitives (sort, compaction)
  - construct index over join keys, add DP noise to it
  - partition the data by keys to fit a partition in the enclave
  - consolidate sparse keys as an optimization
  - do inner join within partition

Detailed Algorithm
Dissertation Prospectus

Secure and Efficient Query Processing in Outsourced Databases

Range Queries [19, 21], Point Queries [21], kNN Queries, JOIN Queries

Dmytro Bogatov
dmytro@bu.edu
Built from 034dbe1e on January 4, 2022

Boston University
Graduate School of Arts and Sciences
Department of Computer Science
REFERENCES


## OPE / ORE schemes

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Primitive usage</th>
<th>Ciphertext size, or state size</th>
<th>Leakage (in addition to inherent total order)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>Encryption</strong></td>
<td><strong>Comparison</strong></td>
<td></td>
</tr>
<tr>
<td>BCLO [1]</td>
<td>$n$ HG</td>
<td>none</td>
<td>$2n$</td>
</tr>
<tr>
<td>CLWW [3]</td>
<td>$n$ PRF</td>
<td>none</td>
<td>$2n$</td>
</tr>
<tr>
<td>Lewi-Wu [13]</td>
<td>$2^n/d$ PRP</td>
<td>$n/2d$ Hash</td>
<td>$n/2d (\lambda + n + 2^{d+1}) + \lambda$</td>
</tr>
<tr>
<td></td>
<td>$2^n d$ (2^d + 1) PRF</td>
<td>$n/2d$ Hash</td>
<td>Most-significant differing block</td>
</tr>
<tr>
<td>CLOZ [2]</td>
<td>$n$ PRF</td>
<td>$n^2$ PPH</td>
<td>$n \cdot h$</td>
</tr>
<tr>
<td>FH-OPE [11]</td>
<td>1 Traversal</td>
<td>3 Traversals</td>
<td>$3 \cdot n \cdot N$</td>
</tr>
</tbody>
</table>
Range query protocols

<table>
<thead>
<tr>
<th>Protocol</th>
<th>I/O requests</th>
<th>Leakage</th>
<th>Communication (result excluded)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Construction</td>
<td>Query</td>
<td></td>
</tr>
<tr>
<td>B+ tree with ORE</td>
<td>(\log_B \frac{N}{B})</td>
<td>(\log_B \frac{N}{B} + \frac{r}{B})</td>
<td>Same as ORE</td>
</tr>
<tr>
<td>Kerschbaum [12]</td>
<td>(\frac{N}{B})</td>
<td>(\log_2 \frac{N}{B} + \frac{r}{B})</td>
<td>Total order</td>
</tr>
<tr>
<td>POPE [14] warm</td>
<td>1</td>
<td>(\log_L \frac{N}{B} + \frac{r}{B})</td>
<td>Partial order</td>
</tr>
<tr>
<td>POPE [14] cold</td>
<td>(\frac{N}{B})</td>
<td>1</td>
<td>Partial order, Fully hiding</td>
</tr>
<tr>
<td>Logarithmic-BRC [4]</td>
<td>—</td>
<td>(r)</td>
<td>Same as SSE</td>
</tr>
<tr>
<td>ORAM</td>
<td>(\log^2 \frac{N}{B})</td>
<td>(\log_2 \frac{N}{B} (\log_B \frac{N}{B} + \frac{r}{B}))</td>
<td>Fully hiding (access pattern)</td>
</tr>
</tbody>
</table>
One of the experimental results

Query stage number of I/O requests

- Uniform distribution
- Normal distribution
- CA public employees dataset

- No encryption
- BCLO, CLWW, FH-OPE
- Lewi-Wu
- CLOZ
- Kerschbaum
- POPE cold
- POPE warm
- Logarithmic
- BRC
- ORAM
Access pattern is a sequence of memory accesses \( y \), where each access consists of the memory location \( o \), read \( r \) or write \( w \) operation and the data \( d \) to be written.

Oblivious RAM (ORAM) is a mechanism that hides the accesses pattern. More formally, ORAM is a protocol between the client \( C \) (who accesses) and the server \( S \) (who stores), with a guarantee that the view of the server is indistinguishable for any two sequences of the same lengths.

\[
|y_1| = |y_2| \\
\text{VIEW}_S(y_1) \approx \text{VIEW}_S(y_2)
\]

For example: Square Root ORAM [8], Hierarchical ORAM [9], Binary-Tree ORAM [16], Interleave Buffer Shuffle Square Root ORAM [22], TP-ORAM [17], Path-ORAM [18] and TaORAM [15]. ORAM incurs at least logarithmic communication overhead in the number of stored records. [9]
\[ \forall x, y, z \in X : \text{DIST}(x, y) < \text{DIST}(x, z) - \beta \implies \text{DIST}(f(x), f(y)) < \text{DIST}(f(x), f(z)) \]

- The scheme is by Riddhi Ghosal and Adam O’Neil [7]
- Key generation: sample at random length multiplier \(s\) and seeds for samplers
- Encrypt: take input vector \(x \in \mathbb{R}^d\)
  - Sample nonce \(n\)
  - Using nonce and seeds, sample a point \(a\) on a \(\beta\)-radius \(d\)-dimensional ball
  - New vector is extended times \(s\) and points to \(a\)
- Decrypt: take encrypted vector \(c \in \mathbb{R}^d\) and nonce \(n\)
  - Do same steps except shrink times \(s\) and remove ball component
The scheme is by Riddhi Ghosal and Adam O’Neil [7]

- **Key generation**: sample at random length multiplier $s$ and seeds for samplers
- **Encrypt**: take input vector $x \in \mathbb{R}^d$
  - Sample nonce $n$
  - Using nonce and seeds, sample a point $a$ on a $\beta$-radius $d$-dimensional ball
  - New vector is extended times $s$ and points to $a$
- **Decrypt**: take encrypted vector $c \in \mathbb{R}^d$ and nonce $n$
  - Do same steps except *shrink* times $s$ and *remove* ball component
∀x, y, z ∈ X : \text{DIST}(x, y) < \text{DIST}(x, z) − \beta \implies \text{DIST}(f(x), f(y)) < \text{DIST}(f(x), f(z))

- The scheme is by Riddhi Ghosal and Adam O’Neil [7]
- **Key generation**: sample at random length multiplier s and seeds for samplers
- **Encrypt**: take input vector \( x \in \mathbb{R}^d \)
  - Sample nonce \( n \)
  - Using nonce and seeds, sample a point \( a \) on a \( \beta \)-radius \( d \)-dimensional ball
  - New vector is extended times s and points to \( a \)
- **Decrypt**: take encrypted vector \( c \in \mathbb{R}^d \) and nonce \( n \)
  - Do same steps except shrink times s and remove ball component
∀x, y, z ∈ X : \( \text{DIST}(x, y) < \text{DIST}(x, z) - \beta \implies \text{DIST}(f(x), f(y)) < \text{DIST}(f(x), f(z)) \)

• The scheme is by Riddhi Ghosal and Adam O’Neil [7]

• **Key generation**: sample at random length multiplier \( s \) and seeds for samplers

• **Encrypt**: take input vector \( x \in \mathbb{R}^d \)
  - Sample nonce \( n \)
  - Using nonce and seeds, sample a point \( a \) on a \( \beta \)-radius \( d \)-dimensional ball
  - New vector is extended times \( s \) and points to \( a \)

• **Decrypt**: take encrypted vector \( c \in \mathbb{R}^d \) and nonce \( n \)
  - Do same steps except *shrink* times \( s \) and *remove* ball component
Component: TREC dataset and FAISS [10]

- Dataset is 8.8M documents represented as vectors of 768 dimensions
  Thanks Hamed Zamani for the dataset
- Query is a 768-dimensional vector asking for $k = 1000$ closest (inner product) documents
- Original document set is a Text REtrieval Conference (TREC) test collection
  set of documents, set of topics (questions), and corresponding set of relevance judgments (right answers)
- FAISS [10]: GPU-enabled library for efficient similarity search and clustering of dense vectors
  Developed and maintained by Facebook AI
- General algorithm: for different $\beta$
  - Encrypt dataset with $\beta$
  - Encrypt queryset with $\beta$
  - Run queries with FAISS
  - Generate TREC metrics (using relevance judgments)
Dmytro Bogatov  
Boston University

Component: TREC dataset and FAISS [10]

- Dataset is 8.8M documents represented as vectors of 768 dimensions
  Thanks Hamed Zamani for the dataset
- Query is a 768-dimensional vector asking for \( k = 1000 \) closest (inner product) documents
- Original document set is a Text REtrieval Conference (TREC) test collection
  set of documents, set of topics (questions), and corresponding set of relevance judgments (right answers)
- FAISS [10]: GPU-enabled library for efficient similarity search and clustering of dense vectors
  Developed and maintained by Facebook AI
- General algorithm: for different \( \beta \)
  - Encrypt dataset with \( \beta \)
  - Encrypt queryset with \( \beta \)
  - Run queries with FAISS
  - Generate TREC metrics (using relevance judgments)
Component: TREC dataset and FAISS [10]

- Dataset is 8.8M documents represented as vectors of 768 dimensions
  Thanks Hamed Zamani for the dataset
- Query is a 768-dimensional vector asking for $k = 1000$ closest (inner product) documents
- Original document set is a Text REtrieval Conference (TREC) test collection
  set of documents, set of topics (questions), and corresponding set of relevance judgments (right answers)
- FAISS [10]: GPU-enabled library for efficient similarity search and clustering of dense vectors
  Developed and maintained by Facebook AI
- General algorithm: for different $\beta$
  - Encrypt dataset with $\beta$
  - Encrypt queryset with $\beta$
  - Run queries with FAISS
  - Generate TREC metrics (using relevance judgments)
Component: TREC dataset and FAISS [10]

- Dataset is 8.8M documents represented as vectors of 768 dimensions
  Thanks Hamed Zamani for the dataset
- Query is a 768-dimensional vector asking for $k = 1000$ closest (inner product) documents
- Original document set is a Text REtrieval Conference (TREC) test collection
  set of documents, set of topics (questions), and corresponding set of relevance judgments (right answers)
- FAISS [10]: GPU-enabled library for efficient similarity search and clustering of dense vectors
  Developed and maintained by Facebook AI
- General algorithm: for different $\beta$
  - Encrypt dataset with $\beta$
  - Encrypt queryset with $\beta$
  - Run queries with FAISS
  - Generate TREC metrics (using relevance judgments)
Component: TREC dataset and FAISS [10]

- Dataset is 8.8M documents represented as vectors of 768 dimensions
  Thanks Hamed Zamani for the dataset
- Query is a 768-dimensional vector asking for $k = 1000$ closest (inner product) documents
- Original document set is a Text REtrieval Conference (TREC) test collection
  set of documents, set of topics (questions), and corresponding set of relevance judgments (right answers)
- FAISS [10]: GPU-enabled library for efficient similarity search and clustering of dense vectors
  Developed and maintained by Facebook AI
- General algorithm: for different $\beta$
  - Encrypt dataset with $\beta$
  - Encrypt queryset with $\beta$
  - Run queries with FAISS
  - Generate TREC metrics (using relevance judgments)
Intermediate results

TREC metrics, result set distance and difference, for running $k$NN search for $\beta \in \{0, 1, \ldots, 50\}$
Oblivious JOINs detailed algorithm

- Construct list $L$ of the form $(k, n_1, \hat{n}_1, n_2, \hat{n}_2)$, with an element per distinct key plus noise
  - $k$ is a join key, $n$ and $\hat{n}$ are real and noisy numbers of records with that key in corresponding input table
  - Noise sampled to a hierarchical sanitizer from a Laplacian distribution

- Client $U$ sends sorted $L$ and hierarchical sanitizer over noise counts to the server $S$
  - Similar to Epsolute, adversary does not learn much from noisy counts

- Server $S$ partitions $L$ by $k$, so that partition size ($\hat{n}_1 + \hat{n}_2$) is bounded and uniform
  - Resulting mapping from keys to partitions $M(k) = i$ can be proven DP

- Consolidate sparse keys: ensure that each bin corresponds to at least $U$ real keys
  - Bin is collection of tuples for which we will do cross-product join

- Obliviously move and pad each bin/partition with dummy records
  - Within each bin the data is sorted by input tables

- For each bin, do cartesian product
Oblivious JOINs detailed algorithm

- Construct list $L$ of the form $(k, n_1, \hat{n}_1, n_2, \hat{n}_2)$, with an element per distinct key plus noise
  $k$ is a join key, $n$ and $\hat{n}$ are real and noisy numbers of records with that key in corresponding input table
  Noise sampled to a hierarchical sanitizer from a Laplacian distribution
- Client $U$ sends sorted $L$ and hierarchical sanitizer over noise counts to the server $S$
  Similar to $\mathcal{E}$psolute, adversary does not learn much from noisy counts
- Server $S$ partitions $L$ by $k$, so that partition size $(\hat{n}_1 + \hat{n}_2)$ is bounded and uniform
  Resulting mapping from keys to partitions $\mathcal{M}(k) = i$ can be proven DP
- Consolidate sparse keys: ensure that each bin corresponds to at least $U$ real keys
  $Bin$ is collection of tuples for which we will do cross-product join
- Obliviously move and pad each bin/partition with dummy records
  Within each bin the data is sorted by input tables
- For each bin, do cartesian product
Oblivious JOINs detailed algorithm

- Construct list $L$ of the form $(k, n_1, \hat{n}_1, n_2, \hat{n}_2)$, with an element per distinct key plus noise
  $k$ is a join key, $n$ and $\hat{n}$ are real and noisy numbers of records with that key in corresponding input table
  Noise sampled to a hierarchical sanitizer from a Laplacian distribution
- Client $U$ sends sorted $L$ and hierarchical sanitizer over noise counts to the server $S$
  Similar to $\text{Epsolute}$, adversary does not learn much from noisy counts
- Server $S$ partitions $L$ by $k$, so that partition size $(\hat{n}_1 + \hat{n}_2)$ is bounded and uniform
  Resulting mapping from keys to partitions $\mathcal{M}(k) = i$ can be proven DP
- Consolidate sparse keys: ensure that each bin corresponds to at least $U$ real keys
  Bin is collection of tuples for which we will do cross-product join
- Obliviously move and pad each bin/partition with dummy records
  Within each bin the data is sorted by input tables
- For each bin, do cartesian product
Oblivious JOINs detailed algorithm

- Construct list $L$ of the form $(k, n_1, \hat{n}_1, n_2, \hat{n}_2)$, with an element per distinct key plus noise
  - $k$ is a join key, $n$ and $\hat{n}$ are real and noisy numbers of records with that key in corresponding input table
  - Noise sampled to a hierarchical sanitizer from a Laplacian distribution
- Client $U$ sends sorted $L$ and hierarchical sanitizer over noise counts to the server $S$
  - Similar to $\mathcal{E}$psolute, adversary does not learn much from noisy counts
- Server $S$ partitions $L$ by $k$, so that partition size $(\hat{n}_1 + \hat{n}_2)$ is bounded and uniform
  - Resulting mapping from keys to partitions $\mathcal{M}(k) = i$ can be proven DP
- Consolidate sparse keys: ensure that each bin corresponds to at least $U$ real keys
  - Bin is collection of tuples for which we will do cross-product join
  - Obliviously move and pad each bin/partition with dummy records
  - Within each bin the data is sorted by input tables
- For each bin, do cartesian product
Oblivious JOINs detailed algorithm

- Construct list $L$ of the form $(k, n_1, \hat{n}_1, n_2, \hat{n}_2)$, with an element per distinct key plus noise
  $k$ is a join key, $n$ and $\hat{n}$ are real and noisy numbers of records with that key in corresponding input table
  Noise sampled to a hierarchical sanitizer from a Laplacian distribution
- Client $U$ sends sorted $L$ and hierarchical sanitizer over noise counts to the server $S$
  Similar to $\varepsilon$psolute, adversary does not learn much from noisy counts
- Server $S$ partitions $L$ by $k$, so that partition size $(\hat{n}_1 + \hat{n}_2)$ is bounded and uniform
  Resulting mapping from keys to partitions $M(k) = i$ can be proven DP
- Consolidate sparse keys: ensure that each bin corresponds to at least $U$ real keys
  Bin is collection of tuples for which we will do cross-product join
- Obliviously move and pad each bin/partition with dummy records
  Within each bin the data is sorted by input tables
- For each bin, do cartesian product
Oblivious JOINs detailed algorithm

- Construct list $L$ of the form $(k, n_1, \hat{n}_1, n_2, \hat{n}_2)$, with an element per distinct key plus noise
  - $k$ is a join key, $n$ and $\hat{n}$ are real and noisy numbers of records with that key in corresponding input table
  - Noise sampled to a hierarchical sanitizer from a Laplacian distribution
- Client $U$ sends sorted $L$ and hierarchical sanitizer over noise counts to the server $S$
  - Similar to $\varepsilon$psolute, adversary does not learn much from noisy counts
- Server $S$ partitions $L$ by $k$, so that partition size $(\hat{n}_1 + \hat{n}_2)$ is bounded and uniform
  - Resulting mapping from keys to partitions $M(k) = i$ can be proven DP
- Consolidate sparse keys: ensure that each bin corresponds to at least $U$ real keys
  - Bin is collection of tuples for which we will do cross-product join
- Obliviously move and pad each bin/partition with dummy records
  - Within each bin the data is sorted by input tables
- For each bin, do cartesian product