Dissertation Prospectus

Secure and Efficient Query Processing in Outsourced Databases

Range Queries [19, 21], Point Queries [21], *k*NN 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`
  - *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

Proposed thesis structure

Dmytro Bogatov, George Kollios, and Leonid Reyzin. A comparative evaluation of order-revealing encryption schemes and secure range-query protocols.
Proceedings of the VLDB Endowment, 12(8):933–947, 2019

Model: snapshot, query type: range

Dmytro Bogatov, Georgios Kellaris, George Kollios, Kobbi Nissim, and Adam O'Neill. εpsolute: Efficiently querying databases while providing differential privacy.
In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security (CCS '2021)*, 2021

Model: persistent, query type: point and range

In-progress: Private kNN queries

Model: snapshot, query type: kNN

In-progress: Oblivious `JOIN` queries

Model: persistent, query type: `JOIN`

Proposed thesis structure

Dmytro Bogatov, George Kollios, and Leonid Reyzin. A comparative evaluation of order-revealing encryption schemes and secure range-query protocols.
Proceedings of the VLDB Endowment, 12(8):933–947, 2019

Model: snapshot, query type: range

Dmytro Bogatov, Georgios Kellaris, George Kollios, Kobbi Nissim, and Adam O'Neill. εpsolute: Efficiently querying databases while providing differential privacy.
In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security (CCS '2021)*, 2021

Model: persistent, query type: point and range

In-progress: Private kNN queries

Model: snapshot, query type: kNN

In-progress: Oblivious `JOIN` queries

Model: persistent, query type: `JOIN`

Proposed thesis structure

Dmytro Bogatov, George Kollios, and Leonid Reyzin. A comparative evaluation of order-revealing encryption schemes and secure range-query protocols.
Proceedings of the VLDB Endowment, 12(8):933–947, 2019

Model: snapshot, query type: range

Dmytro Bogatov, Georgios Kellaris, George Kollios, Kobbi Nissim, and Adam O'Neill. εpsolute: Efficiently querying databases while providing differential privacy.
In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security (CCS '2021)*, 2021

Model: persistent, query type: point and range

In-progress: Private kNN queries

Model: snapshot, query type: kNN

In-progress: Oblivious `JOIN` queries

Model: persistent, query type: `JOIN`

Proposed thesis structure

Dmytro Bogatov, George Kollios, and Leonid Reyzin. A comparative evaluation of order-revealing encryption schemes and secure range-query protocols.
Proceedings of the VLDB Endowment, 12(8):933–947, 2019

Model: snapshot, query type: range

Dmytro Bogatov, Georgios Kellaris, George Kollios, Kobbi Nissim, and Adam O'Neill. εpsolute: Efficiently querying databases while providing differential privacy.
In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security (CCS '2021)*, 2021

Model: persistent, query type: point and range

In-progress: Private kNN queries

Model: snapshot, query type: kNN

In-progress: Oblivious `JOIN` queries

Model: persistent, query type: `JOIN`

Proposed thesis structure

Dmytro Bogatov, George Kollios, and Leonid Reyzin. A comparative evaluation of order-revealing encryption schemes and secure range-query protocols.
Proceedings of the VLDB Endowment, 12(8):933–947, 2019

Model: snapshot, query type: range

Dmytro Bogatov, Georgios Kellaris, George Kollios, Kobbi Nissim, and Adam O'Neill. εpsolute: Efficiently querying databases while providing differential privacy.
In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security (CCS '2021)*, 2021

Model: persistent, query type: point and range

In-progress: Private kNN queries

Model: snapshot, query type: kNN

In-progress: Oblivious `JOIN` queries

Model: persistent, query type: `JOIN`

Dmytro Bogatov, Angelo De Caro, Kaoutar Elkhiyaoui, and Björn Tackmann. Anonymous transactions with revocation and auditing in hyperledger fabric.
In *International Conference on Cryptology and Network Security*. Springer, 2021

A comparative evaluation of order-revealing encryption schemes and secure range-query protocols [19]

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**

» ORE table » Protocols table » Plot

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**

⏩ ORE table ⏩ Protocols table ⏩ Plot

εpsolute: Efficiently Querying Databases While Providing Differential Privacy [21]

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 $\mathcal{O}(\log n)$ ▸▸ ORAM definition
- 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 $\mathcal{O}(\log n)$ ▸▸ ORAM definition
- 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 $\mathcal{O}(\log n)$ ▸▸ ORAM definition
- 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 (ϵ, δ)-differentially private if for all $\mathcal{D}_1 \sim \mathcal{D}_2 \in \mathcal{X}^n$, and for all subsets $\mathcal{O}$ of the output space of A,

$$\Pr\left[\mathrm{A}\left(\mathcal{D}_1\right) \in \mathcal{O}\right] \leq \exp(\epsilon) \cdot \Pr\left[\mathrm{A}\left(\mathcal{D}_2\right) \in \mathcal{O}\right] + \delta \ .$$

How to make sense of it?

- Differential Privacy is a property of an algorithm
  What about ϵ and δ?
- 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 (ϵ, δ)-differentially private if for all $\mathcal{D}_1 \sim \mathcal{D}_2 \in \mathcal{X}^n$, and for all subsets $\mathcal{O}$ of the output space of A,

$$\Pr\left[\mathrm{A}\left(\mathcal{D}_1\right) \in \mathcal{O}\right] \leq \exp(\epsilon) \cdot \Pr\left[\mathrm{A}\left(\mathcal{D}_2\right) \in \mathcal{O}\right] + \delta \ .$$

How to make sense of it?

- Differential Privacy is a property of an algorithm
  What about ϵ and δ?
- 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 (ϵ, δ)-differentially private if for all $\mathcal{D}_1 \sim \mathcal{D}_2 \in \mathcal{X}^n$, and for all subsets $\mathcal{O}$ of the output space of A,

$$\Pr\left[\mathrm{A}\left(\mathcal{D}_1\right) \in \mathcal{O}\right] \leq \exp(\epsilon) \cdot \Pr\left[\mathrm{A}\left(\mathcal{D}_2\right) \in \mathcal{O}\right] + \delta \ .$$

How to make sense of it?

- Differential Privacy is a property of an algorithm
  What about ϵ and δ?
- 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 (ϵ, δ)-differentially private if for all $\mathcal{D}_1 \sim \mathcal{D}_2 \in \mathcal{X}^n$, and for all subsets $\mathcal{O}$ of the output space of A,

$$\Pr\left[\mathrm{A}\left(\mathcal{D}_1\right) \in \mathcal{O}\right] \leq \exp(\epsilon) \cdot \Pr\left[\mathrm{A}\left(\mathcal{D}_2\right) \in \mathcal{O}\right] + \delta \ .$$

How to make sense of it?

- Differential Privacy is a property of an algorithm
  What about ϵ and δ?
- How to construct such an algorithm?
  Laplace Perturbation Method!
- What if negative value is sampled?
  Cannot truncate one side, must shift entire distribution

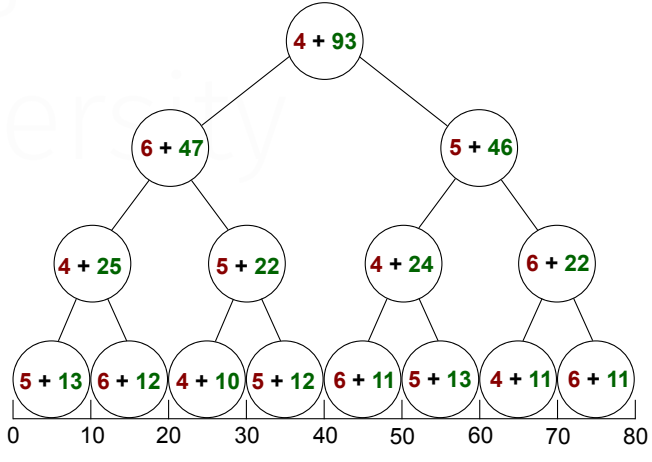
Definition (Differential Privacy, adapted from [5, 6])

A randomized algorithm A is (ϵ, δ)-differentially private if for all $\mathcal{D}_1 \sim \mathcal{D}_2 \in \mathcal{X}^n$, and for all subsets $\mathcal{O}$ of the output space of A,

$$\Pr\left[\mathrm{A}\left(\mathcal{D}_1\right) \in \mathcal{O}\right] \leq \exp(\epsilon) \cdot \Pr\left[\mathrm{A}\left(\mathcal{D}_2\right) \in \mathcal{O}\right] + \delta \ .$$

How to make sense of it?

- Differential Privacy is a property of an algorithm
  What about ϵ and δ?
- 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 Π is (ϵ, δ)-computationally differentially private (a.k.a. CDP-ODB) if for every polynomial time distinguishing adversary $\mathcal{A}$, for every neighboring databases $\mathcal{D} \sim \mathcal{D}'$, and for every query sequence $q_1, \ldots, q_m \in \mathcal{Q}^m$ where $m = \text{poly}(\lambda)$,

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

the probability is over the randomness of the distinguishing adversary $\mathcal{A}$ and the protocol Π.

Note:

- Entire view of the adversary is DP-protected
- Implies protection against **communication volume** and **access pattern** leakages
- Query sequence $q_1, \ldots, q_m \in \mathcal{Q}^m$ 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 Π is (ϵ, δ)-computationally differentially private (a.k.a. CDP-ODB) if for every polynomial time distinguishing adversary $\mathcal{A}$, for every neighboring databases $\mathcal{D} \sim \mathcal{D}'$, and for every query sequence $q_1, \ldots, q_m \in \mathcal{Q}^m$ where $m = \text{poly}(\lambda)$,

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

the probability is over the randomness of the distinguishing adversary $\mathcal{A}$ and the protocol Π.

Note:

- Entire view of the adversary is DP-protected
- Implies protection against **communication volume** and **access pattern** leakages
- Query sequence $q_1, \ldots, q_m \in \mathcal{Q}^m$ 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 $q_1, \ldots, q_m \in \mathcal{Q}^m$ 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 $q_1, \ldots, q_m \in \mathcal{Q}^m$ fixed?

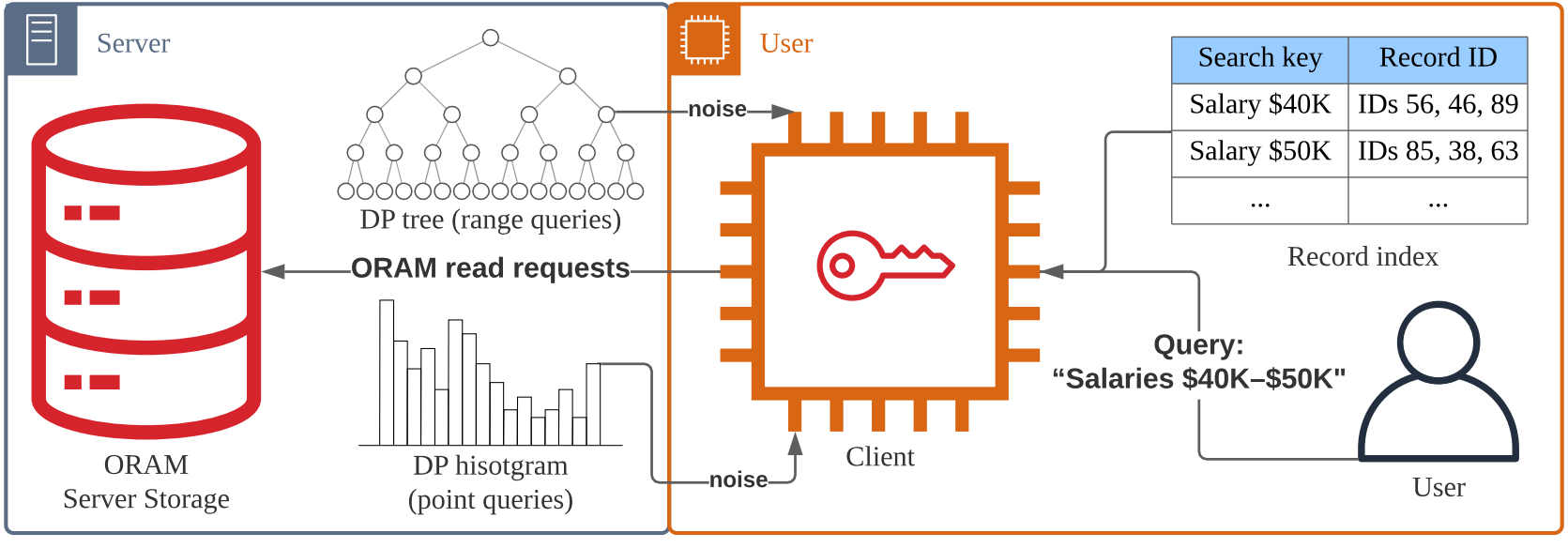
- Suppose neighboring medical databases differ in one record with a rare diagnosis "Alzheimer's disease"
- A medical professional, who is **a user, not an adversary** queries the database
  - for that diagnosis first
    `SELECT name FROM patients WHERE condition = 'ALZ'`
  - if there is a record, she 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

On impossibility of adaptive queries

Why is the query sequence $q_1, \ldots, q_m \in \mathcal{Q}^m$ 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

Single-Threaded $\mathcal{E}$psolute protocol



Parallel $\mathcal{E}$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 $\mathcal{DS}$?

No-γ method: $\mathcal{DS}$ per ORAM

- Composition of disjoint datasets: take max ϵ
- 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

γ-method: shared $\mathcal{DS}$

- 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

- 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 $\mathcal{DS}$?

No-γ method: $\mathcal{DS}$ per ORAM

- Composition of disjoint datasets: take max ϵ
- 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

γ-method: shared $\mathcal{DS}$

- 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

- 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 $\mathcal{DS}$?

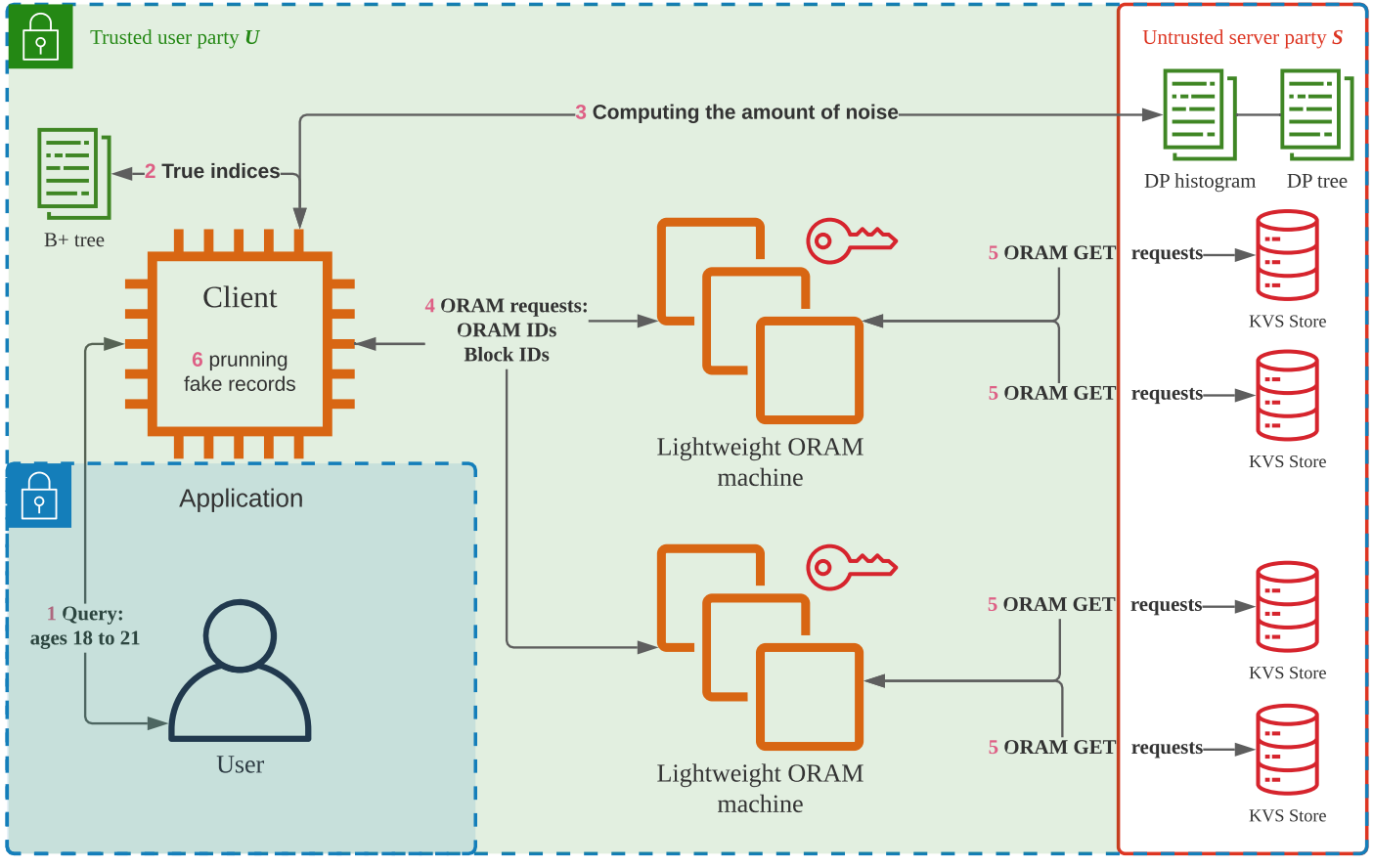
No-γ method: $\mathcal{DS}$ per ORAM

- Composition of disjoint datasets: take max ϵ
- 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

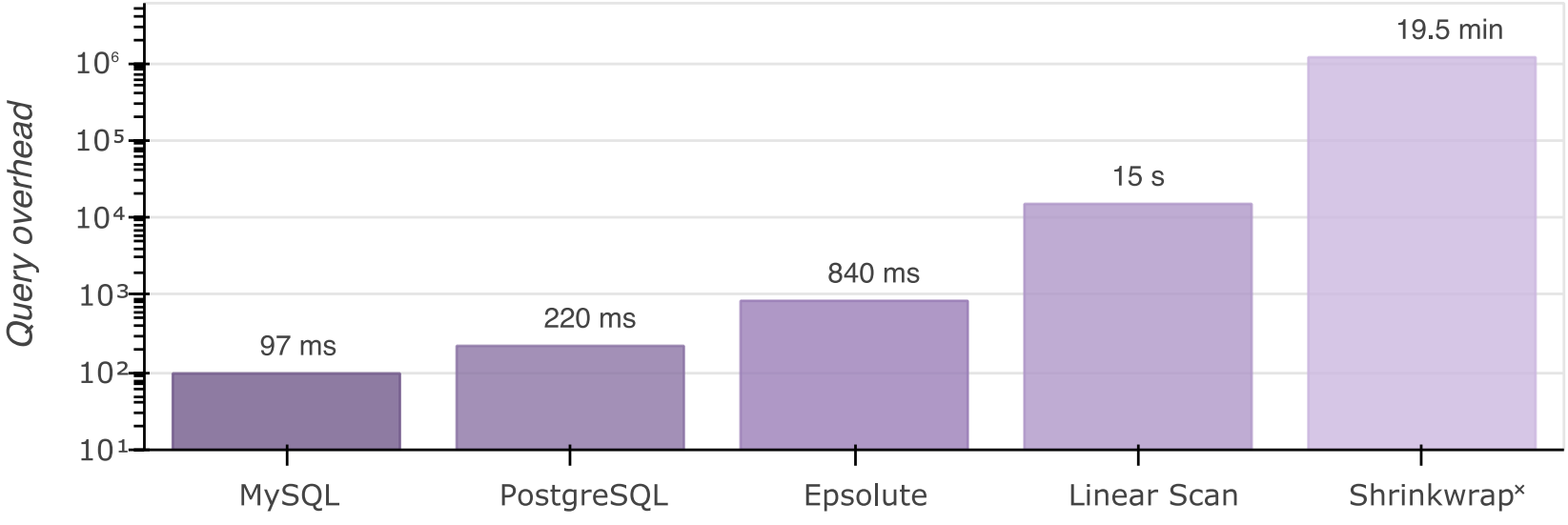
γ-method: shared $\mathcal{DS}$

- 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 $\mathcal{E}$psolute diagram (with improvements)

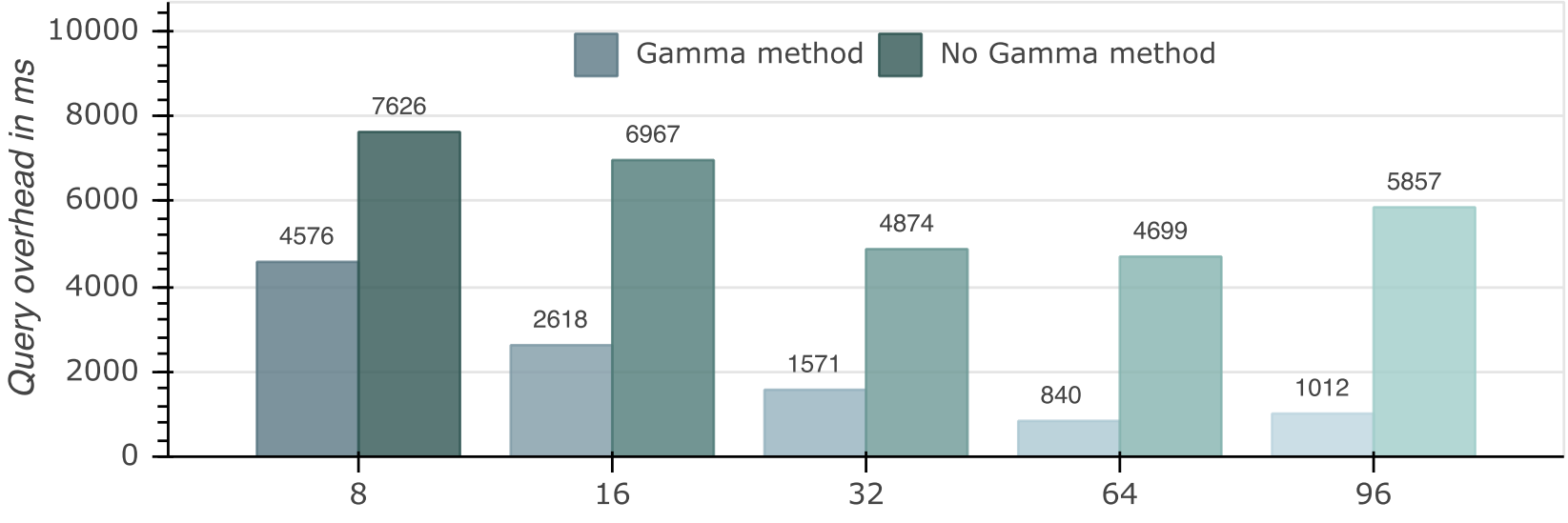


Experiments: against other mechanisms



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

Experiments: scalability



Scalability measurements for Π_{γ} (shared $\mathcal{DS}$) and $\Pi_{no-\gamma}$ ($\mathcal{DS}$ per ORAM)

Work-in-progress: private kNN queries

General idea

- Model: snapshot, query type: kNN 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

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

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

DCPE TREC and FAISS Intermediate results plot

General idea

- Model: snapshot, query type: kNN 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

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

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

DCPE TREC and FAISS Intermediate results plot

General idea

- Model: snapshot, query type: kNN 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

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

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

DCPE · TREC and FAISS · Intermediate results plot

General idea

- Model: snapshot, query type: kNN 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

$$\forall x, y, z \in \mathbb{X} : \textsc{Dist}(x, y) < \textsc{Dist}(x, z) - \beta \implies \textsc{Dist}(f(x), f(y)) < \textsc{Dist}(f(x), f(z))$$

- 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: kNN 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

$$\forall x, y, z \in \mathbb{X} : \textsc{Dist}(x, y) < \textsc{Dist}(x, z) - \beta \implies \textsc{Dist}(f(x), f(y)) < \textsc{Dist}(f(x), f(z))$$

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

⏩ DCPE ⏩ TREC and FAISS ⏩ Intermediate results plot

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 `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 `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 `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

Dissertation Prospectus

Secure and Efficient Query Processing in Outsourced Databases

Range Queries [19, 21], Point Queries [21], *k*NN 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

[1] Alexandra Boldyreva, Nathan Chenette, Younho Lee, and Adam O'Neill. Order-preserving symmetric encryption. In *Advances in Cryptology - EUROCRYPT 2009*, pages 224–241. Springer Berlin Heidelberg, 2009.

[2] David Cash, Feng-Hao Liu, Adam O'Neill, Mark Zhandry, and Cong Zhang. Parameter-hiding order revealing encryption. In *Advances in Cryptology – ASIACRYPT 2018*, pages 181–210. Springer International Publishing, 2018.

[3] Nathan Chenette, Kevin Lewi, Stephen A. Weis, and David J. Wu. Practical order-revealing encryption with limited leakage. In *Fast Software Encryption*, pages 474–493. Springer Berlin Heidelberg, 2016.

[4] Ioannis Demertzis, Stavros Papadopoulos, Odysseas Papapetrou, Antonios Deligiannakis, and Minos Garofalakis. Practical private range search revisited. In *Proceedings of the 2016 International Conference on Management of Data*, pages 185–198, 2016.

[5] Cynthia Dwork, Krishnaram Kenthapadi, Frank McSherry, Ilya Mironov, and Moni Naor. Our data, ourselves: Privacy via distributed noise generation. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 486–503. Springer, 2006.

[6] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. Calibrating noise to sensitivity in private data analysis. In *Theory of cryptography conference*, pages 265–284. Springer, 2006.

[7] Georg Fuchsbauer, Riddhi Ghosal, Nathan Hauke, and Adam O'Neill. Approximate distance-comparison-preserving symmetric encryption. Cryptology ePrint Archive, Report 2021/1666, 2021. https://ia.cr/2021/1666.

[8] Oded Goldreich. Towards a theory of software protection and simulation by oblivious rams. In *Proceedings of the nineteenth annual ACM symposium on Theory of computing*, pages 182–194, 1987.

[9] Oded Goldreich and Rafail Ostrovsky. Software protection and simulation on oblivious rams. *Journal of the ACM (JACM)*, 43(3):431–473, 1996.

[10] Jeff Johnson, Matthijs Douze, and Hervé Jégou. Billion-scale similarity search with gpus. *arXiv preprint arXiv:1702.08734*, 2017.

[11] Florian Kerschbaum. Frequency-hiding order-preserving encryption. In *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security*, pages 656–667. ACM, 2015.

[12] Florian Kerschbaum and Anselme Tueno. An efficiently searchable encrypted data structure for range queries. In *Computer Security – ESORICS 2019*, pages 344–364. Springer International Publishing, 2019.

[13] Kevin Lewi and David J. Wu. Order-revealing encryption: New constructions, applications, and lower bounds. pages 1167–1178. ACM, 2016.

[14] Daniel S. Roche, Daniel Apon, Seung Geol Choi, and Arkady Yerukhimovich. POPE: Partial order preserving encoding. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 1131–1142. ACM, 2016.

[15] Cetin Sahin, Victor Zakhary, Amr El Abbadi, Huijia Lin, and Stefano Tessaro. Taostore: Overcoming asynchronicity in oblivious data storage. In *2016 IEEE Symposium on Security and Privacy (SP)*, pages 198–217. IEEE, 2016.

[16] Elaine Shi, T-H Hubert Chan, Emil Stefanov, and Mingfei Li. Oblivious ram with $o(\log^3 n)$ worst-case cost. In *International Conference on The Theory and Application of Cryptology and Information Security*, pages 197–214. Springer, 2011.

[17] Emil Stefanov, Elaine Shi, and Dawn Xiaodong Song. Towards practical oblivious RAM. In *Network and Distributed System Security Symposium (NDSS)*, 2012.

[18] Emil Stefanov, Marten van Dijk, Elaine Shi, Christopher Fletcher, Ling Ren, Xiangyao Yu, and Srinivas Devadas. Path oram: An extremely simple oblivious ram protocol. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer Communications Security*, pages 299–310. ACM, 2013.

[19] **Dmytro Bogatov**, George Kollios, and Leonid Reyzin. A comparative evaluation of order-revealing encryption schemes and secure range-query protocols. *Proceedings of the VLDB Endowment*, 12(8):933–947, 2019.

[20] **Dmytro Bogatov**, Angelo De Caro, Kaoutar Elkhiyaoui, and Björn Tackmann. Anonymous transactions with revocation and auditing in hyperledger fabric. In *International Conference on Cryptology and Network Security*. Springer, 2021.

[21] **Dmytro Bogatov**, Georgios Kellaris, George Kollios, Kobbi Nissim, and Adam O'Neill. $\mathcal{E}$psolute: Efficiently querying databases while providing differential privacy. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security (CCS '2021)*, 2021.

[22] Dong Xie, Guanru Li, Bin Yao, Xuan Wei, Xiaokui Xiao, Yunjun Gao, and Minyi Guo. Practical private shortest path computation based on oblivious storage. In *2016 IEEE 32nd International Conference on Data Engineering (ICDE)*, pages 361–372. IEEE, 2016.

Appendix

OPE / ORE schemes

Scheme	Primitive usage		Ciphertext size, or state size	Leakage (in addition to inherent total order)
	Encryption	Comparison		
BCLO [1]	n **HG**	none	$2n$	$\approx$ **Top half of the bits**
CLWW [3]	n PRF	none	$2n$	**Most-significant differing bit**
Lewi-Wu [13]	$2n/d$ **PRP** $2\frac{n}{d}\left(2^d+1\right)$ PRF $\frac{n}{d}2^d$ Hash	$\frac{n}{2d}$ Hash	$\frac{n}{d}\left(\lambda+n+2^{d+1}\right)+\lambda$	Most-significant differing block
CLOZ [2]	n PRF n PPH 1 PRP	n^2 **PPH**	$n \cdot h$	Equality pattern of most-significant differing bit
FH-OPE [11]	1 Traversal	3 Traversals	$3 \cdot n \cdot N$	Insertion order

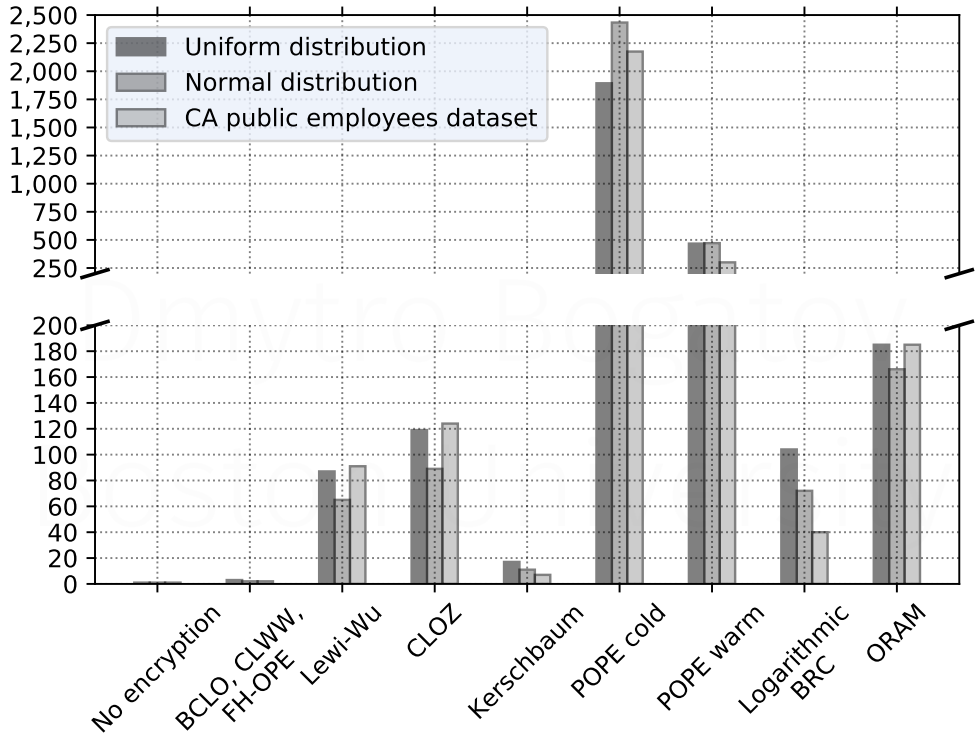
Back to ORE

Range query protocols

Protocol	I/O requests		Leakage	Communication (result excluded)	
	Construction	Query		Construction	Query
B+ tree with ORE	$\log_B \frac{N}{B}$	$\log_B \frac{N}{B} + \frac{r}{B}$	**Same as ORE**	1	1
Kerschbaum [12]	$\frac{N}{B}$	$\log_2 \frac{N}{B} + \frac{r}{B}$	**Total order**	$\log_2 N$	$\log_2 N$
POPE [14] warm	1	$\log_L \frac{N}{B} + \frac{r}{B}$	**Partial order**	1	$\log_L N$
POPE [14] cold		N/B	Fully hiding		N
Logarithmic-BRC [4]	—	r	Same as SSE	—	$\log_2 N$
ORAM	$\log^2 \frac{N}{B}$	$\log_2 \frac{N}{B} \left(\log_B \frac{N}{B} + \frac{r}{B}\right)$	Fully hiding (access pattern)	$\log^2 \frac{N}{B}$	$\log^2 \frac{N}{B}$

◂ Back to ORE

One of the experimental results



Query stage number of I/O requests

Back to ORE

Access pattern and ORAM

Access pattern is a sequence of memory accesses $\mathbf{y}$, where each access consists of the memory *location* o, read $\mathbf{r}$ or write $\mathbf{w}$ *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 $\mathcal{C}$ (who accesses) and the server $\mathcal{S}$ (who stores), with a guarantee that the view of the server is indistinguishable for any two sequences of the same lengths.

$$|\mathbf{y}_1| = |\mathbf{y}_2|$$
$$\text{VIEW}_{\mathcal{S}}(\mathbf{y}_1) \overset{c}{\approx} \text{VIEW}_{\mathcal{S}}(\mathbf{y}_2)$$

ORAM protocol

	Client $\mathcal{C}$		Server $\mathcal{S}$
1 :			
2 :	$\mathbf{y} = (\mathbf{r}, i, \perp)\|_{i=1}^{5}$		
3 :	(client state)	ORAM $(\mathbf{y})$ ⟷	(server state)
4 :	$\{d_1, d_2, d_3, d_4, d_5\}$		

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]

Back to $\mathcal{E}$psolute

Component: DCPE

$$\forall x, y, x \in \mathbb{X} : \textsc{Dist}(x, y) < \textsc{Dist}(x, z) - \beta \implies \textsc{Dist}(f(x), f(y)) < \textsc{Dist}(f(x), f(z))$$

- 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 β-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

◂ Back to kNN

Component: DCPE

$$\forall x, y, x \in \mathbb{X} : \textsc{Dist}(x, y) < \textsc{Dist}(x, z) - \beta \implies \textsc{Dist}(f(x), f(y)) < \textsc{Dist}(f(x), f(z))$$

- 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 β-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

◂ Back to kNN

$$\forall x, y, x \in \mathbb{X} : \textsc{Dist}(x, y) < \textsc{Dist}(x, z) - \beta \implies \textsc{Dist}(f(x), f(y)) < \textsc{Dist}(f(x), f(z))$$

- 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 β-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

◂ Back to kNN

Component: DCPE

$$\forall x, y, x \in \mathbb{X} : \textsc{Dist}(x, y) < \textsc{Dist}(x, z) - \beta \implies \textsc{Dist}(f(x), f(y)) < \textsc{Dist}(f(x), f(z))$$

- 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 β-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

◂ Back to kNN

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 **T**ext **RE**trieval **C**onference (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 β
  - Encrypt dataset with β
  - Encrypt queryset with β
  - Run queries with FAISS
  - Generate TREC metrics (using relevance judgments)

Back to kNN

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 **T**ext **RE**trieval **C**onference (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 β
  - Encrypt dataset with β
  - Encrypt queryset with β
  - Run queries with FAISS
  - Generate TREC metrics (using relevance judgments)

Back to kNN

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 **T**ext **RE**trieval **C**onference (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 β
  - Encrypt dataset with β
  - Encrypt queryset with β
  - Run queries with FAISS
  - Generate TREC metrics (using relevance judgments)

Back to *k*NN

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 **T**ext **RE**trieval **C**onference (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 β
  - Encrypt dataset with β
  - Encrypt queryset with β
  - Run queries with FAISS
  - Generate TREC metrics (using relevance judgments)

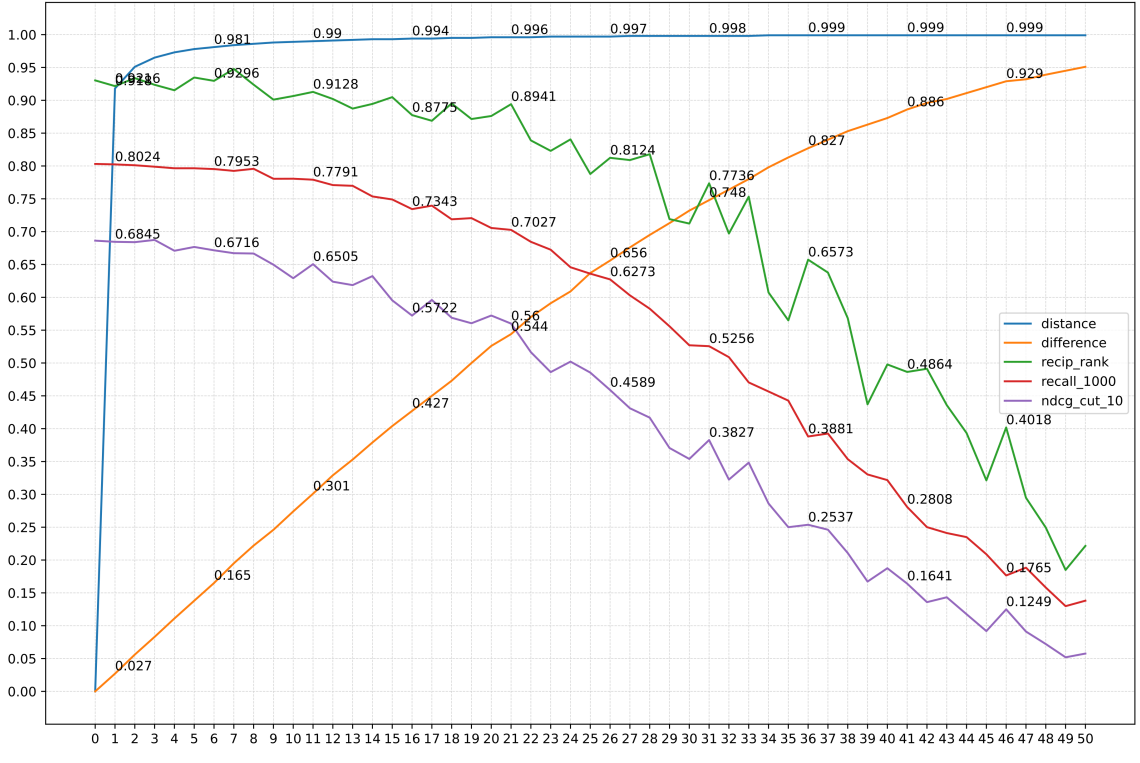
Back to *k*NN

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 **T**ext **RE**trieval **C**onference (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 β
  - Encrypt dataset with β
  - Encrypt queryset with β
  - Run queries with FAISS
  - Generate TREC metrics (using relevance judgments)

Back to *k*NN

Intermediate results



TREC metrics, result set distance and difference, for running kNN search for $\beta \in \{0, 1, \ldots, 50\}$

◂ Back to kNN

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 $\mathcal{U}$ sends sorted L and hierarchical sanitizer over noise counts to the server $\mathcal{S}$
  Similar to $\mathcal{E}$psolute, adversary does not learn much from noisy counts
- Server $\mathcal{S}$ 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**

◂ Back to Oblivious Joins

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 $\mathcal{U}$ sends sorted L and hierarchical sanitizer over noise counts to the server $\mathcal{S}$
  Similar to $\mathcal{E}$psolute, adversary does not learn much from noisy counts
- Server $\mathcal{S}$ 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**

◂ Back to Oblivious Joins

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 $\mathcal{U}$ sends sorted L and hierarchical sanitizer over noise counts to the server $\mathcal{S}$
  Similar to $\mathcal{E}$psolute, adversary does not learn much from noisy counts
- Server $\mathcal{S}$ 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**

◂ Back to Oblivious Joins

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 $\mathcal{U}$ sends sorted L and hierarchical sanitizer over noise counts to the server $\mathcal{S}$
  Similar to $\mathcal{E}$psolute, adversary does not learn much from noisy counts
- Server $\mathcal{S}$ 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**

◂ Back to Oblivious Joins

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 $\mathcal{U}$ sends sorted L and hierarchical sanitizer over noise counts to the server $\mathcal{S}$
  Similar to $\mathcal{E}$psolute, adversary does not learn much from noisy counts
- Server $\mathcal{S}$ 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**

◂ Back to Oblivious Joins

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 $\mathcal{U}$ sends sorted L and hierarchical sanitizer over noise counts to the server $\mathcal{S}$
  Similar to $\mathcal{E}$psolute, adversary does not learn much from noisy counts
- Server $\mathcal{S}$ 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**

◂ Back to Oblivious Joins