Secure and Efficient Query Processing in Outsourced Databases

Range Queries \cite{BKR19, Bog+21}, Point Queries \cite{Bog+21}, \(k\)NN Queries \cite{BKOZ22}

Dmytro Bogatov
dmytro@bu.edu
Built from 3c92001c on February 16, 2023
Boston University
Graduate School of Arts and Sciences
Department of Computer Science
INTRODUCTION
Motivation and overview

- With vast amounts of data, organizations choose to use cloud
- **Challenge:** solutions must be both **secure** and **efficient**
- Security models for an outsourced database system
  - **Snapshot** adversary: steal the hard drive and RAM snapshot
  - **Persistent** adversary: continuously monitor the entire server
- 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`
Motivation and overview

- With vast amounts of data, organizations choose to use cloud
- **Challenge:** solutions must be both *secure* and *efficient*
- Security models for an outsourced database system
  - **Snapshot** adversary: steal the hard drive and RAM snapshot
  - **Persistent** adversary: continuously monitor the entire server
- 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
Motivation and overview

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

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

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

• 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
Motivation and overview

• With vast amounts of data, organizations choose to use cloud
• **Challenge:** solutions must be both **secure** and **efficient**
• Security models for an outsourced database system
  • **Snapshot** adversary: steal the hard drive and RAM snapshot
  • **Persistent** adversary: continuously monitor the entire server
• 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**
Motivation and overview

- With vast amounts of data, organizations choose to use cloud
- **Challenge:** solutions must be both **secure** and **efficient**
- Security models for an outsourced database system
  - **Snapshot** adversary: steal the hard drive and RAM snapshot
  - **Persistent** adversary: continuously monitor the entire server
- 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`
Motivation and overview

- With vast amounts of data, organizations choose to use cloud
- **Challenge**: solutions must be both **secure** and **efficient**
- Security models for an outsourced database system
  - **Snapshot** adversary: steal the hard drive and RAM snapshot
  - **Persistent** adversary: continuously monitor the entire server
- 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`
Works published during the Ph.D. program


Works published during the Ph.D. program


My work

Works published during the Ph.D. program


My work

Works published during the Ph.D. program


A COMPARATIVE EVALUATION OF ORDER-REVEALING ENCRYPTION SCHEMES AND SECURE RANGE-QUERY PROTOCOLS [BKR19]
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 schemes and protocols
- 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
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 schemes and protocols
- 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 [Bog+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 $O(\log n)$
  - 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!
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!
Definition (Differential Privacy, adapted from [Dwo+06; DMNS06])

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 \( A \),

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

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 [Dwo+06; DMNS06])

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 [Dwo+06; DMNS06])

A randomized algorithm $A$ is $(\epsilon, \delta)$-differentially private if for all $D_1 \sim D_2 \in X^n$, 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 [Dwo+06; DMNS06])

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 [Dwo+06; DMNS06])

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)

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
- $\text{negl}(\lambda)$ needed for the computational (as opposed to information-theoretical) DP definition
Definition (Computationally Differentially Private Outsourced Database System)

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 $m = \text{poly}(\lambda)$,

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

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
- Negl($\lambda$) needed for the computational (as opposed to information-theoretical) DP definition
Single-Threaded $\epsilon$psolute protocol

Server:
- ORAM Storage
- ORAM read requests
- ORAM (point queries)
- DP tree (range queries)

User:
- DP histogram
- Key
- Noise

Client:
- Client
- Noise
- Query: "Salaries $40K-50K"

Record index:
- Search key
- Record ID
- Salary $40K: IDs 56, 46, 89
- Salary $50K: IDs 85, 38, 63
- ...
Parallel $\varepsilon$psolute

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

$\Pi_{\text{separate}}$: separate sanitizer $\mathcal{DS}$ per ORAM
  - 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, all ORAMs are processing $r + mf$ records in parallel

$\Pi_{\text{shared}}$: shared sanitizer $\mathcal{DS}$ for all ORAMs
  - Same number of total records per ORAM
  - Generated noise is larger than $f$ (say, $\alpha f$), but split among $m$ ORAMs
  - That is, all ORAMs are processing $r + \alpha f$ records in parallel

$\Pi_{\text{shared}}$ wins if $\alpha < m$, which it is for almost all values of $m$ ($m \geq 4$).
Parallel $\mathcal{E}$psolute

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

$\Pi_{\text{separate}}$: separate sanitizer $\mathcal{D}S$ per ORAM
  - 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, all ORAMs are processing $r + mf$
    records in parallel

$\Pi_{\text{shared}}$: shared sanitizer $\mathcal{D}S$ for all ORAMs
  - Same number of total records per ORAM
  - Generated noise is larger than $f$ (say, $\alpha f$), but
    split among $m$ ORAMs
  - That is, all ORAMs are processing $r+\alpha f$ records
    in parallel

$\Pi_{\text{shared}}$ wins if $\alpha < m$, which it is for almost all values of $m$ ($m \geq 4$)
Parallel $\varepsilon$psolute

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

$\Pi_{\text{separate}}$: separate sanitizer $\mathcal{DS}$ per ORAM
  - 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, all ORAMs are processing $r + mf$ records in parallel

$\Pi_{\text{shared}}$: shared sanitizer $\mathcal{DS}$ for all ORAMs
  - Same number of total records per ORAM
  - Generated noise is larger than $f$ (say, $\alpha f$), but split among $m$ ORAMs
  - That is, all ORAMs are processing $r + \alpha f$ records in parallel

$\Pi_{\text{shared}}$ wins if $\alpha < m$, which it is for almost all values of $m$ ($m \geq 4$)
Parallel $\varepsilon$psolute

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

$\Pi_{\text{separate}}$: separate sanitizer $\mathcal{DS}$ per ORAM
- 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, all ORAMs are processing $r + mf$ records in parallel

$\Pi_{\text{shared}}$: shared sanitizer $\mathcal{DS}$ for all ORAMs
- Same number of total records per ORAM
- Generated noise is larger than $f$ (say, $\alpha f$), but split among $m$ ORAMs
- That is, all ORAMs are processing $r + \alpha f$ records in parallel

$\Pi_{\text{shared}}$ wins if $\alpha < m$, which it is for almost all values of $m$ ($m \gtrapprox 4$)
Parallel $\xi$psolute diagram (with improvements)

1. Query: ages 18 to 21
2. True indices
3. Computing the amount of noise
4. ORAM reqs: ORAM IDs Block IDs
5. ORAM GET requests
6. pruning fake records

Client

Trustx user party $U$

Untrusted server party $S$

Application

User

Lightweight ORAM machine

KVS Store

KVS Store

KVS Store

KVS Store

KVS Store

DP histogram

DP tree

DP tree
Experiments: scalability

Scalability measurements for $\Pi_{\text{shared}}$ and $\Pi_{\text{separate}}$ ($DS$ is a DP sanitizer)
Experiments: against other mechanisms

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

- MySQL: 97 ms
- PostgreSQL: 220 ms
- Epsolute: 840 ms
- Linear Scan: 15 s
- Shrinkwrap*: 19.5 min
$k$-anon: Secure Similarity Search in Outsourced Databases [BKOZ22]
General idea

- Model: **snapshot**, query type: **$k$NN** in arbitrary dimensions
  - Nearest-neighbor search needs definitions of **object** and **distance**
    - **Object** can be 2D/3D location, or a document embedding (high-dimensional vector)
    - **Distance** can be a Euclidean distance or inner (dot) product distance
  - **Query** then can be “5 closest restaurants” or “3 most similar documents”

- Our approach is to apply an **approximate property-preserving encryption** on objects
  - Query protocol is then similar to OPE / ORE
  - Existing nearest-neighbor search algorithms then work naturally

- Study how accuracy of search and efficiency of attacks drop with higher security
• Model: snapshot, query type: $k$NN in arbitrary dimensions

• Nearest-neighbor search needs definitions of object and distance
  Object can be 2D/3D location, or a document embedding (high-dimensional vector)
  Distance can be a Euclidean distance or inner (dot) product distance
  Query then can be “5 closest restaurants” or “3 most similar documents”

• Our approach is to apply an approximate property-preserving encryption on objects
  Query protocol is then similar to OPE / ORE
  Existing nearest-neighbor search algorithms then work naturally

• Study how accuracy of search and efficiency of attacks drop with higher security
General idea

- Model: snapshot, query type: $k$NN in arbitrary dimensions
- Nearest-neighbor search needs definitions of object and distance
  - Object can be 2D/3D location, or a document embedding (high-dimensional vector)
  - Distance can be a Euclidean distance or inner (dot) product distance
  - Query then can be “5 closest restaurants” or “3 most similar documents”
- Our approach is to apply an approximate property-preserving encryption on objects
  - Query protocol is then similar to OPE / ORE
  - Existing nearest-neighbor search algorithms then work naturally
- Study how accuracy of search and efficiency of attacks drop with higher security
General idea

- Model: snapshot, query type: $k$NN in arbitrary dimensions
- Nearest-neighbor search needs definitions of object and distance
  - Object can be 2D/3D location, or a document embedding (high-dimensional vector)
  - Distance can be a Euclidean distance or inner (dot) product distance
  - Query then can be “5 closest restaurants” or “3 most similar documents”
- Our approach is to apply an approximate property-preserving encryption on objects
  - Query protocol is then similar to OPE / ORE
  - Existing nearest-neighbor search algorithms then work naturally
- Study how accuracy of search and efficiency of attacks drop with higher security
∀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))

If distance between x and z is larger than the distance between x and y by more than \( \beta \), then the encryption of z will be further than the encryption of y from the encryption of x.

Distance Comparison Preserving Encryption scheme [FGHO21]
Setup and query protocols: for given $\beta$
- Generate encryption key
- Encrypt dataset and queries set with $\beta$
- Run queries using conventional nearest-neighbor search (e.g., FAISS)
- Report search accuracy metrics

TREC 2020 dataset is 8.8M documents embedded with fine-tuned BERT (768 dimensions)
Thanks Hamed Zamani for the dataset

Query is a 768-dimensional embedding asking for $k = 1000$ closest documents
TREC has a set of documents, a set of topics (questions), and relevance judgments (right answers)

We report result set distance and difference, and ranking quality Recall, MRR and nDCG
Set distance and difference measure pure $k$NN accuracy
Recall, MRR and nDCG report ranking quality using relevance, common in information retrieval literature
Search accuracy and TREC dataset

- Setup and query protocols: for given $\beta$
  - Generate encryption key
  - Encrypt dataset and queries set with $\beta$
  - Run queries using conventional nearest-neighbor search (e.g., FAISS)
  - Report search accuracy metrics
- TREC 2020 dataset is 8.8M documents embedded with fine-tuned BERT (768 dimensions)
  Thanks Hamed Zamani for the dataset
- Query is a 768-dimensional embedding asking for $k = 1000$ closest documents
- TREC has a set of documents, a set of topics (questions), and relevance judgments (right answers)
- We report result set distance and difference, and ranking quality Recall, MRR and nDCG
  Set distance and difference measure pure $k$NN accuracy
  Recall, MRR and nDCG report ranking quality using relevance, common in information retrieval literature
Search accuracy and TREC dataset

- Setup and query protocols: for given $\beta$
  - Generate encryption key
  - Encrypt dataset and queries set with $\beta$
  - Run queries using conventional nearest-neighbor search (e.g., FAISS)
  - Report search accuracy metrics

- TREC 2020 dataset is 8.8M documents embedded with fine-tuned BERT (768 dimensions)
  Thanks Hamed Zamani for the dataset

- Query is a 768-dimensional embedding asking for $k = 1000$ closest documents
  TREC has a set of documents, a set of topics (questions), and relevance judgments (right answers)

- We report result set distance and difference, and ranking quality Recall, MRR and nDCG
  Set distance and difference measure pure $k$NN accuracy
  Recall, MRR and nDCG report ranking quality using relevance, common in information retrieval literature
Setup and query protocols: for given $\beta$
- Generate encryption key
- Encrypt dataset and queries set with $\beta$
- Run queries using conventional nearest-neighbor search (e.g., FAISS)
- Report search accuracy metrics

TREC 2020 dataset is 8.8M documents embedded with fine-tuned BERT (768 dimensions)
Thanks Hamed Zamani for the dataset

Query is a 768-dimensional embedding asking for $k = 1000$ closest documents

TREC has a set of documents, a set of topics (questions), and relevance judgments (right answers)

We report result set distance and difference, and ranking quality Recall, MRR and nDCG
Set distance and difference measure pure $k$NN accuracy
Recall, MRR and nDCG report ranking quality using relevance, common in information retrieval literature
Search accuracy results

Rank quality metrics, result set distance and difference for $\beta \in \{0, 1, \ldots, 50\}$
Black-box inversion attack [SR20]

- ML model trains on document-embedding pairs and predicts a set of words from embedding
  - Model is an LSTM trained for 30 epochs
  - Original attack used BookCorpus [Zhu+15] dataset, but we will use TREC
- We evaluate the attack on encrypted embeddings
  - We also add plaintext and random embeddings for the baselines
  - Public model: adversary can use the embedding model, therefore, trains on plaintexts
  - Private model: adversary can only use the entire system, therefore, trains on ciphertexts
  - In both cases the model predicts the words from the encrypted embedding
- We measure precision, recall and F$_1$ score along with the percent of stop-words
  - Stop-words are common words like “a”, “the”, pronouns, even punctuation and digits
Black-box inversion attack [SR20]

- ML model trains on document-embedding pairs and predicts a set of words from embedding. Model is an LSTM trained for 30 epochs.
- Original attack used BookCorpus [Zhu+15] dataset, but we will use TREC.
- We evaluate the attack on encrypted embeddings.
  - We also add plaintext and random embeddings for the baselines.
  - **Public model**: adversary can use the embedding model, therefore, trains on plaintexts.
  - **Private model**: adversary can only use the entire system, therefore, trains on ciphertexts.
  - In both cases the model predicts the words from the encrypted embedding.
- We measure precision, recall and F1 score along with the percent of stop-words.
  Stop-words are common words like "a", "the", pronouns, even punctuation and digits.
Black-box inversion attack [SR20]

- ML model trains on document-embedding pairs and predicts a set of words from embedding
  Model is an LSTM trained for 30 epochs
  Original attack used BookCorpus [Zhu+15] dataset, but we will use TREC
- We evaluate the attack on encrypted embeddings
  - We also add plaintext and random embeddings for the baselines
  - **Public model**: adversary can use the embedding model, therefore, trains on plaintexts
  - **Private model**: adversary can only use the entire system, therefore, trains on ciphertexts
  - In both cases the model predicts the words from the encrypted embedding
- We measure precision, recall and F1 score along with the percent of stop-words
  Stop-words are common words like “a”, “the”, pronouns, even punctuation and digits
Attack performance results: public model (trained on plaintext)
Attack performance results: private model (trained on ciphertext)

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Precision</th>
<th>Recall</th>
<th>$F_1$ score</th>
<th>% of non-stop-words</th>
</tr>
</thead>
<tbody>
<tr>
<td>Encrypted with $\beta = 0.0$</td>
<td>38.84</td>
<td>27.64</td>
<td>32.30</td>
<td>2.31</td>
</tr>
<tr>
<td>Encrypted with $\beta = 4.0$</td>
<td>36.28</td>
<td>26.61</td>
<td>30.70</td>
<td>3.21</td>
</tr>
<tr>
<td>Random embeddings</td>
<td>36.07</td>
<td>26.61</td>
<td>30.62</td>
<td>0.00</td>
</tr>
</tbody>
</table>
Tradeoff between search accuracy and attack performance (public model, trained on plaintext)
CONCLUSIONS
Future work directions

• Focus on practicality and reproducibility!
  • Property-preserving encryption is practical [BKR19; BKOZ22]
    May not be ideally-secure, and does not have to be
    Benchmark the scheme and quantify the leakage
  • Hardware gets cheaper, consider “heavy” primitives and protocols [Bog+21]
    ORAM, homomorphic encryption, garbled circuits, zero-knowledge proofs, etc
    Performance may be acceptable with optimizations, specialized hardware and parallelization
  • More database query types in outsourced model
    JOIN, GROUP BY, AGGREGATE, custom predicates, etc
Future work directions

• Focus on practicality and reproducibility!

• Property-preserving encryption is practical [BKR19; BKOZ22]
  May not be ideally-secure, and does not have to be
  Benchmark the scheme and quantify the leakage

• Hardware gets cheaper, consider “heavy” primitives and protocols [Bog+21]
  ORAM, homomorphic encryption, garbled circuits, zero-knowledge proofs, etc
  Performance may be acceptable with optimizations, specialized hardware and parallelization

• More database query types in outsourced model
  JOIN, GROUP BY, AGGREGATE, custom predicates, etc
Future work directions

• **Focus on practicality and reproducibility!**

• Property-preserving encryption is practical [BKR19; BKOZ22]
  May not be ideally-secure, and does not have to be
  Benchmark the scheme and quantify the leakage

• Hardware gets cheaper, consider “heavy” primitives and protocols [Bog+21]
  ORAM, homomorphic encryption, garbled circuits, zero-knowledge proofs, etc
  Performance may be acceptable with optimizations, specialized hardware and parallelization

• More database query types in outsourced model
  JOIN, GROUP BY, AGGREGATE, custom predicates, etc
Future work directions

- **Focus on practicality and reproducibility!**
- Property-preserving encryption is practical [BKR19; BKOZ22]
  May not be ideally-secure, and does not have to be
  Benchmark the scheme and quantify the leakage
- Hardware gets cheaper, consider “heavy” primitives and protocols [Bog+21]
  ORAM, homomorphic encryption, garbled circuits, zero-knowledge proofs, etc
  Performance may be acceptable with optimizations, specialized hardware and parallelization
- More database query types in outsourced model
  JOIN, GROUP BY, AGGREGATE, custom predicates, etc
Acknowledgements
Secure and Efficient Query Processing in Outsourced Databases

Range Queries [BKR19; Bog+21], Point Queries [Bog+21], kNN Queries [BKOZ22]

Dmytro Bogatov
dmytro@bu.edu

Built from 3c92001c on February 16, 2023

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


[Xie+16] 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)*. IEEE. 2016, pp. 361–372.


<table>
<thead>
<tr>
<th>Scheme</th>
<th>Primitive usage</th>
<th>Ciphertext size, Leakage</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Encryption</td>
<td>Comparison</td>
</tr>
<tr>
<td>BCLO [BCLO09]</td>
<td>$n$ HG</td>
<td>none</td>
</tr>
<tr>
<td>CLWW [CLWW16]</td>
<td>$n$ PRF</td>
<td>none</td>
</tr>
<tr>
<td>Lewi-Wu [LW16]</td>
<td>$\frac{2n}{d}$ PRP</td>
<td>$\frac{n}{2^d}$ Hash</td>
</tr>
<tr>
<td>CLOZ [Cas+18]</td>
<td>$n$ PRF</td>
<td>$n^2$ PPH</td>
</tr>
<tr>
<td></td>
<td>$n$ PPH</td>
<td>$n^2$ PPH</td>
</tr>
<tr>
<td></td>
<td>1 PRP</td>
<td></td>
</tr>
<tr>
<td>FH-OPE [Ker15]</td>
<td>1 Traversal</td>
<td>3 Traversals</td>
</tr>
</tbody>
</table>

Table 1: [BKR19, Table 1]. Primitive usage by OPE / ORE schemes. Ordered by security rank — most secure below. $n$ is the input length in bits, $d$ is a block size for Lewi-Wu [LW16] scheme, $\lambda$ is a PRF output size, $N$ is a total data size, HG is a hyper-geometric distribution sampler, PPH is a property-preserving hash with $h$-bit outputs built with bilinear maps and **bolded** are weak points of the schemes.
## 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>Construction</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 [KT19]</td>
<td>$\frac{N}{B}$</td>
<td>$\log_2 \frac{N}{B} + \frac{r}{B}$</td>
<td>Total order</td>
</tr>
<tr>
<td>POPE [RACY16] warm</td>
<td>1</td>
<td>$\log_L \frac{N}{B} + \frac{r}{B}$</td>
<td>Partial order</td>
</tr>
<tr>
<td>POPE [RACY16] cold</td>
<td>1</td>
<td>$\log_L \frac{N}{B} + \frac{r}{B}$</td>
<td>Fully hiding</td>
</tr>
<tr>
<td>Logarithmic-BRC [Dem+16]</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>

**Table 2:** [BKR19, Tables 2]. Performance of the range query protocols. Ordered by security rank — most secure below. $N$ is a total data size, $B$ is an I/O page size, $L$ is a POPE tree branching factor, $r$ is the result size in records and **bolded** are weak points of the protocols.
One of the experimental results

![Bar chart showing the number of I/O requests for different datasets and encryption methods. The x-axis represents various encryption methods such as No encryption, BCLO, CLWW, FH-OPE, Lewi-Wu, CLOZ, Kerschbaum, POPE cold, POPE warm, Logarithmic, BRC, ORAM. The y-axis represents the number of I/O requests ranging from 0 to 2,500. The chart includes bars for Uniform distribution, Normal distribution, and CA public employees dataset. The query stage number of I/O requests is shown at the bottom.]
Access pattern and 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.

$$|y_1| = |y_2|$$

$$\text{VIEW}_S(y_1) \approx \text{VIEW}_S(y_2)$$

**ORAM protocol**

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

Square Root ORAM [Gol87], Hierarchical ORAM [GO96], Binary-Tree ORAM [SCSL11], Interleave Buffer Shuffle Square Root ORAM [Xie16], TP-ORAM [SSS12], PathORAM [Ste13] and TaORAM [Sah16]. ORAM incurs at least logarithmic overhead in the number of stored records. [GO96]
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 (and 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 (and 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
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 (and 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
### Algorithm 1: Distance Comparison Preserving Encryption, adapted from [FGHO21, Algorithm 2]

#### KEYGEN($1^\lambda$, $\mathbb{S}$)

1: $s \leftarrow \mathbb{S}$
2: $k \leftarrow \{0, 1\}^\lambda$
3: return $(s, k)$

#### ENC($s$, $k$, $\vec{m}$)

1: $n \leftarrow \{0, 1\}^\lambda$
2: $\text{coins}_n || \text{coins}_u \leftarrow \text{PRF}(k, n)$
3: $\vec{n} \leftarrow \mathbb{S} \text{NORMAL}(0, l_d; \text{coins}_n)$
4: $u \leftarrow \mathbb{S} \text{UNIFORM}(0, 1; \text{coins}_u)$
5: $x \leftarrow \frac{s \beta}{4} \cdot \sqrt{u}$
6: $\vec{\delta} \leftarrow \frac{\vec{n}}{\|\vec{n}\|} \cdot x$
7: $\vec{c} \leftarrow s \cdot \vec{m} + \vec{\delta}$
8: return $\vec{c}$

#### DEC($s$, $k$, ($\vec{c}$, $n$))

1: $\text{coins}_n || \text{coins}_u \leftarrow \text{PRF}(k, n)$
2: $\vec{n} \leftarrow \mathbb{S} \text{NORMAL}(0, l_d; \text{coins}_n)$
3: $u \leftarrow \mathbb{S} \text{UNIFORM}(0, 1; \text{coins}_u)$
4: $x \leftarrow \frac{s \beta}{4} \cdot \sqrt{u}$
5: $\vec{\delta} \leftarrow \frac{\vec{n}}{\|\vec{n}\|} \cdot x$
6: $\vec{m} \leftarrow \frac{\vec{c} - \vec{\delta}}{s}$
7: return $\vec{m}$