Epsolute: Efficiently Querying Databases While Providing Differential Privacy

Differential Privacy, ORAM, differential obliviousness, sanitizers

[42] DOI: 10.1145/3460120.3484786

Dmytro Bogatov, Georgios Kellaris, George Kollios, Kobbi Nissim, Adam O’Neill
dmytro@bu.edu, kellaris@bu.edu, gkollios@cs.bu.edu,
kobbi.nissim@georgetown.edu, adamo@cs.umass.edu

Built from 078878e4 on October 17, 2021

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

• With vast amounts of data, organizations choose to use cloud solutions
• These solutions need to be both efficient and secure
  • Recent attacks on access pattern (AP) [8, 9, 16, 18, 19, 21, 26, 29, 31] and communication volume (CV) [21, 30, 31, 36, 40]
  • Existing solutions may be insufficient:
    • protection against snapshot adversary does not account for AP and CV
      CryptDB [6], Arx [37], Seabed [22] and SisoSPIR [20]
    • enclaves like SGX are still uncommon and limited in memory
      Cipherbase [11], HardIDX [25], StealthDB [38], EnclaveDB [33], ObliDB [35], Opaque [27] and Oblix [32]
    • other solutions protect either from one of AP or CV, or use linear scan and full padding
      Cryptε [41], Shrinkwrap [28], SEAL [39] and PINED-RQ [34]
  • Epsolute: most secure and practical range- and point-query engine in the outsourced database model, that protects both AP and CV using Differential Privacy, while not relying on TEE, linear scan or full padding
Motivation

- With vast amounts of data, organizations choose to use cloud solutions
- These solutions need to be both efficient and secure
- Recent attacks on access pattern (AP) \([8, 9, 16, 18, 19, 21, 26, 29, 31]\) and communication volume (CV) \([21, 30, 31, 36, 40]\)

- Existing solutions may be insufficient:
  - protection against snapshot adversary does not account for AP and CV
    CryptDB \([6]\), Arx \([37]\), Seabed \([22]\) and SisoSPIR \([20]\)
  - enclaves like SGX are still uncommon and limited in memory
    Cipherbase \([11]\), HardIDX \([25]\), StealthDB \([38]\), EnclaveDB \([33]\), ObliDB \([35]\), Opaque \([27]\) and Oblix \([32]\)
  - other solutions protect either from one of AP or CV, or use linear scan and full padding
    Crypt\(\epsilon\) \([41]\), Shrinkwrap \([28]\), SEAL \([39]\) and PINED-RQ \([34]\)

- \(\mathcal{E}\)psolute: most secure and practical range- and point-query engine in the outsourced database model, that protects both AP and CV using Differential Privacy, while not relying on TEE, linear scan or full padding
Motivation

- With vast amounts of data, organizations choose to use cloud solutions
- These solutions need to be both efficient and secure
- Recent attacks on access pattern (AP) [8, 9, 16, 18, 19, 21, 26, 29, 31] and communication volume (CV) [21, 30, 31, 36, 40]
- Existing solutions may be insufficient:
  - protection against snapshot adversary does not account for AP and CV CryptDB [6], Arx [37], Seabed [22] and SisoSPIR [20]
  - enclaves like SGX are still uncommon and limited in memory Cipherbase [11], HardIDX [25], StealthDB [38], EnclaveDB [33], ObliDB [35], Opaque [27] and Oblix [32]
  - other solutions protect either from one of AP or CV, or use linear scan and full padding Cryptε [41], Shrinkwrap [28], SEAL [39] and PINED-RQ [34]
  - Epsolute: most secure and practical range- and point-query engine in the outsourced database model, that protects both AP and CV using Differential Privacy, while not relying on TEE, linear scan or full padding
Motivation

• With vast amounts of data, organizations choose to use cloud solutions
• These solutions need to be both efficient and secure
• Recent attacks on access pattern (AP) [8, 9, 16, 18, 19, 21, 26, 29, 31] and communication volume (CV) [21, 30, 31, 36, 40]
• Existing solutions may be insufficient:
  • protection against snapshot adversary does not account for AP and CV CryptDB [6], Arx [37], Seabed [22] and SisoSPIR [20]
  • enclaves like SGX are still uncommon and limited in memory Cipherbase [11], HardIDX [25], StealthDB [38], EnclaveDB [33], ObliDB [35], Opaque [27] and Oblix [32]
  • other solutions protect either from one of AP or CV, or use linear scan and full padding Cryptɛ [41], Shrinkwrap [28], SEAL [39] and PINED-RQ [34]
  • Epsolute: most secure and practical range- and point-query engine in the outsourced database model, that protects both AP and CV using Differential Privacy, while not relying on TEE, linear scan or full padding
Motivation

• With vast amounts of data, organizations choose to use cloud solutions
• These solutions need to be both efficient and secure
• Recent attacks on access pattern (AP) [8, 9, 16, 18, 19, 21, 26, 29, 31] and communication volume (CV) [21, 30, 31, 36, 40]

• Existing solutions may be insufficient:
  • protection against snapshot adversary does not account for AP and CV
    CryptDB [6], Arx [37], Seabed [22] and SisoSPIR [20]
  • enclaves like SGX are still uncommon and limited in memory
    Cipherbase [11], HardIDX [25], StealthDB [38], EnclaveDB [33], ObliDB [35], Opaque [27] and Oblix [32]
  • other solutions protect either from one of AP or CV, or use linear scan and full padding
    Cryptɛ [41], Shrinkwrap [28], SEAL [39] and PINED-RQ [34]
• Ïpsolute: most secure and practical range- and point-query engine in the outsourced database model, that protects both AP and CV using Differential Privacy, while not relying on TEE, linear scan or full padding
Motivation

• With vast amounts of data, organizations choose to use cloud solutions
• These solutions need to be both efficient and secure
• Recent attacks on access pattern (AP) \([8, 9, 16, 18, 19, 21, 26, 29, 31]\) and communication volume (CV) \([21, 30, 31, 36, 40]\)

• Existing solutions may be insufficient:
  • protection against snapshot adversary does not account for AP and CV
    CryptDB \([6]\), Arx \([37]\), Seabed \([22]\) and SisoSPIR \([20]\)
  • enclaves like SGX are still uncommon and limited in memory
    Cipherbase \([11]\), HardIDX \([25]\), StealthDB \([38]\), EnclaveDB \([33]\), ObliDB \([35]\), Opaque \([27]\) and Oblix \([32]\)
  • other solutions protect either from one of AP or CV, or use linear scan and full padding
    Crypt\(\epsilon\) \([41]\), Shrinkwrap \([28]\), SEAL \([39]\) and PINED-RQ \([34]\)
  • \(\epsilon\)psolute: most secure and practical range- and point-query engine in the outsourced database model, that protects both AP and CV using Differential Privacy, while not relying on TEE, linear scan or full padding
Database

A database is modeled as a set of tuples consisting of record payload, record unique ID, and record search key (i.e., the indexed attribute).

Query

A query is a predicate that is evaluated on a search key. Evaluating a query on a database results in all records whose search keys satisfy that query.

Protocols

User $U$ and server $S$ are stateful. In setup, $U$ receives a (plaintext) database, $S$ has no input. $S$ outputs data structure $DS$, $U$ has no output. In query, $U$ receives a query, $S$ receives the data structure $DS$. $U$ outputs the result of evaluating the query on a database, $S$ has no output.

Correctness

For correctness, we require that for any database and query, it holds that running the protocols yields for $U$ the correct output with overwhelming probability.
Database

A database is modeled as a set of tuples consisting of record payload, record unique ID, and record search key (i.e., the indexed attribute).

Query

A query is a predicate that is evaluated on a search key. Evaluating a query on a database results in all records whose search keys satisfy that query.

Protocols

User $U$ and server $S$ are stateful. In setup, $U$ receives a (plaintext) database, $S$ has no input. $S$ outputs data structure $DS$, $U$ has no output. In query, $U$ receives a query, $S$ receives the data structure $DS$. $U$ outputs the result of evaluating the query on a database, $S$ has no output.

Correctness

For correctness, we require that for any database and query, it holds that running the protocols yields for $U$ the correct output with overwhelming probability.
Database

A database is modeled as a set of tuples consisting of record payload, record unique ID, and record search key (i.e., the indexed attribute).

Query

A query is a predicate that is evaluated on a search key. Evaluating a query on a database results in all records whose search keys satisfy that query.

Protocols

User $U$ and server $S$ are stateful. In setup, $U$ receives a (plaintext) database, $S$ has no input. $S$ outputs data structure $DS$, $U$ has no output. In query, $U$ receives a query, $S$ receives the data structure $DS$. $U$ outputs the result of evaluating the query on a database, $S$ has no output.

Correctness

For correctness, we require that for any database and query, it holds that running the protocols yields for $U$ the correct output with overwhelming probability.
Database

A database is modeled as a set of tuples consisting of record payload, record unique ID, and record search key (i.e., the indexed attribute).

Query

A query is a predicate that is evaluated on a search key. Evaluating a query on a database results in all records whose search keys satisfy that query.

Protocols

User $U$ and server $S$ are stateful. In $\textit{setup}$, $U$ receives a (plaintext) database, $S$ has no input. $S$ outputs data structure $DS$, $U$ has no output. In $\textit{query}$, $U$ receives a query, $S$ receives the data structure $DS$. $U$ outputs the result of evaluating the query on a database, $S$ has no output.

Correctness

For correctness, we require that for any database and query, it holds that running the protocols yields for $U$ the correct output with overwhelming probability.
Differential Privacy and Sanitization

**Definition (Differential Privacy, adapted from [3, 4])**

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 $A$,

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

**Theorem (Laplace Perturbation Algorithm (LPA), adapted Theorem 1 from [3])**

An algorithm $A$ that adds independently generated noise from a zero-mean Laplace distribution with scale $\lambda = \frac{\text{sensitivity of query}}{\epsilon}$ to each coordinate of a query result, satisfies $\epsilon$-differential privacy.

**Definition (Differentially Private Sanitizer, informal)**

An $(\epsilon, \delta, \alpha, \beta)$-differentially private sanitizer is a pair of algorithms $(A, B)$ such that:

- $A$ is $(\epsilon, \delta)$-differentially private, and
- on input a dataset, $A$ outputs a data structure $DS$ such that with probability $1 - \beta$ for all queries, $B(DS, q)$ is within $\alpha$ of a real query result.
Differential Privacy and Sanitization

Definition (Differential Privacy, adapted from [3, 4])

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 $A$,

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

Theorem (Laplace Perturbation Algorithm (LPA), adapted Theorem 1 from [3])

An algorithm $A$ that adds independently generated noise from a zero-mean Laplace distribution with scale $\lambda = \frac{\text{sensitivity of query}}{\epsilon}$ to each coordinate of a query result, satisfies $\epsilon$-differential privacy.

Definition (Differentially Private Sanitizer, informal)

An $(\epsilon, \delta, \alpha, \beta)$-differentially private sanitizer is a pair of algorithms $(A, B)$ such that:

- $A$ is $(\epsilon, \delta)$-differentially private, and
- on input a dataset, $A$ outputs a data structure $DS$ such that with probability $1 - \beta$ for all queries, $B(DS, q)$ is within $\alpha$ of a real query result.
Definition (Differential Privacy, adapted from [3, 4])

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,

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

Theorem (Laplace Perturbation Algorithm (LPA), adapted Theorem 1 from [3])

An algorithm A that adds independently generated noise from a zero-mean Laplace distribution with scale \(\lambda = \frac{\text{sensitivity of query}}{\epsilon}\) to each coordinate of a query result, satisfies \(\epsilon\)-differential privacy.

Definition (Differentially Private Sanitizer, informal)

An \((\epsilon, \delta, \alpha, \beta)\)-differentially private sanitizer is a pair of algorithms \((A, B)\) such that:

- A is \((\epsilon, \delta)\)-differentially private, and
- on input a dataset, A outputs a data structure \(DS\) such that with probability \(1 - \beta\) for all queries, \(B(DS, q)\) is within \(\alpha\) of a real query result.
Answering point and range queries with differential privacy. $N$ is a domain size. Omitting the dependency on $\epsilon$ and $\delta$, shown are values of $\alpha$.

<table>
<thead>
<tr>
<th></th>
<th>Point Query</th>
<th>Range Query</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pure DP</td>
<td>$\Theta(\log N)$ [15]</td>
<td>$\Theta(\log N)$ [5, 13]</td>
</tr>
<tr>
<td>Approximate DP</td>
<td>$\mathcal{O}(1)$ [12]</td>
<td>$\mathcal{O}(2^{\log^* N})$ [12, 17]</td>
</tr>
</tbody>
</table>

**Theorem (Composition)**

Let $A_1, \ldots, A_r$ be mechanisms, such that each $A_i$ provides $\epsilon_i$-differential privacy. Let $D_1, \ldots, D_r$ be pairwise non-disjoint (resp., disjoint) datasets. Let $A$ be another mechanism that executes $A_1(D_1), \ldots, A_r(D_r)$ using independent randomness for each $A_i$, and returns their outputs. Then, mechanism $A$ is $(\sum_{i=1}^r \epsilon_i)$-differentially private (resp., $(\max_{i=1}^r \epsilon_i)$-differentially private).
Answering point and range queries with differential privacy. $N$ is a domain size. Omitting the dependency on $\epsilon$ and $\delta$, shown are values of $\alpha$.

<table>
<thead>
<tr>
<th></th>
<th>Point Query</th>
<th>Range Query</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pure DP</td>
<td>$\Theta(\log N)$ [15]</td>
<td>$\Theta(\log N)$ [5, 13]</td>
</tr>
<tr>
<td>Approximate DP</td>
<td>$O(1)$ [12]</td>
<td>$O(2^{\log^* N})$ [12, 17]</td>
</tr>
</tbody>
</table>

**Theorem (Composition)**

Let $A_1, \ldots, A_r$ be mechanisms, such that each $A_i$ provides $\epsilon_i$-differential privacy. Let $D_1, \ldots, D_r$ be pairwise non-disjoint (resp., disjoint) datasets. Let $A$ be another mechanism that executes $A_1(D_1), \ldots, A_r(D_r)$ using independent randomness for each $A_i$, and returns their outputs. Then, mechanism $A$ is $(\sum_{i=1}^r \epsilon_i)$-differentially private (resp., $(\max_{i=1}^r \epsilon_i)$-differentially private).
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. ORAM is a protocol between $C$ (who accesses) and $S$ (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. $y = (r, i, ⊥)_{i=1}^5$

3. (client state)

4. $\{d_1, d_2, d_3, d_4, d_5\}$

   (server state)

For example: Square Root ORAM [1], Hierarchical ORAM [2], Binary-Tree ORAM [7], Interleave Buffer Shuffle Square Root ORAM [24], TP-ORAM [10], Path-ORAM [14] and TaORAM [23]. ORAM incurs at least logarithmic overhead in the number of stored records. [2]
Contributions: Model, Single-Threaded and Parallel EPSOLUTE
Differentially Private Outsourced Database System

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

We say that an outsourced database system $\Pi$ is ($\epsilon, \delta$)-computationally differentially private (a.k.a. CDP-OBD) 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
- $\text{negl}(\lambda)$ accounts for the computational (as opposed to theoretical) DP definition
Example

• Suppose neighboring medical databases differ in one record with a rare diagnosis “Alzheimer’s disease”
• A medical professional queries the database
  • for that diagnosis first (point query)
    SELECT name FROM patients WHERE condition = 'ALZ'
  • if there is a record, she queries the senior patients next (range query)
    SELECT name FROM patients WHERE age >= 65
  • otherwise she queries the general population, resulting in more records
    SELECT name FROM patients
• Efficient system cannot return nearly the same number of records in both cases, thus, the adversary can distinguish
• In setup protocol, lookup index over the records is stored on the client $\mathcal{U}$ and all records are stored on the server $\mathcal{S}$ via ORAM (record ID is the ORAM location).
• In query protocol, the ORAM locations are retrieved from the index, the total (real + noise) records number is fetched from the sanitized $\mathcal{DS}$, records are retrieved via ORAM.
• Refer to the full paper [42] for formal description and security proof.
Sanitizers for point and range queries

Point queries

- Use the LPA method as the sanitizer to ensure pure differential privacy
- For every histogram bin, draw from $\text{LAPLACE} \left( \alpha, \frac{1}{\epsilon} \right)$

Range queries

- Use the aggregate tree method as the sanitizer, a $k$-ary tree over domain ($k = 16$)
- A leaf node holds the number of records falling into each bin plus some noise;
- A parent node holds sum of the leaf values in the range covered by this node, plus noise;
- Sanitizer’s output is the best-range cover;
- For every node, draw from $\text{LAPLACE} \left( \alpha, \frac{\log_k N}{\epsilon} \right)$

Set $\alpha$ to the smallest such that if drawn $N$ times, values are positive with probability $1 - \beta$.

$$\alpha = \left\lceil \frac{- \ln \left( 2 - 2^{\sqrt{1 - \beta}} \epsilon \right)}{\epsilon} \right\rceil$$

$$\alpha = \left\lceil \frac{- \ln \left( 2 - 2^{\text{nodes} / \sqrt{1 - \beta}} \cdot \log_k N \right)}{\epsilon} \right\rceil$$
Sanitizers for point and range queries

Point queries

- Use the LPA method as the sanitizer to ensure pure differential privacy
- For every histogram bin, draw from \( \text{LAPLACE} \left( \alpha, \frac{1}{\epsilon} \right) \)

Range queries

- Use the aggregate tree method as the sanitizer, a \( k \)-ary tree over domain \( (k = 16) \)
  - a leaf node holds the number of records falling into each bin plus some noise;
  - a parent node holds sum of the leaf values in the range covered by this node, plus noise;
  - sanitizer’s output is the best-range cover;
- For every node, draw from \( \text{LAPLACE} \left( \alpha, \frac{\log_k N}{\epsilon} \right) \)

Set \( \alpha \) to the smallest such that if drawn \( N \) times, values are positive with probability \( 1 - \beta \).

\[
\alpha = \left\lceil -\frac{\ln \left( 2 - 2^{\frac{1}{\sqrt{1 - \beta}}} \right)}{\epsilon} \right\rceil
\]

\[
\alpha = \left\lceil -\frac{\ln \left( 2 - 2^{\frac{\text{nodes}}{\sqrt{1 - \beta}}} \right) \cdot \log_k N}{\epsilon} \right\rceil
\]
Storage efficiency is defined as the sum of the bit-lengths of the records in a database relative to the bit-length of a corresponding encrypted database. Communication efficiency is defined as the sum of the lengths of the records in bits whose search keys satisfy the query relative to the actual number of bits sent back as the result of a query. Efficiency is defined as \((a_1, a_2)\), where \(a_1\) is a multiplicative term and \(a_2\) is the additive one.

Epsolute’s storage efficiency is \((\mathcal{O}(1), 0)\). Communication efficiencies for different query and DP types are shown.

<table>
<thead>
<tr>
<th></th>
<th>Point Query</th>
<th>Range Query</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pure DP</td>
<td>((\mathcal{O}(\log n), \mathcal{O}(\log N \log n)))</td>
<td>((\mathcal{O}(\log n), \mathcal{O}(\log N \log n)))</td>
</tr>
<tr>
<td>Approximate DP</td>
<td>((\mathcal{O}(\log n), \mathcal{O}(\log n)))</td>
<td>((\mathcal{O}(\log n), \mathcal{O}(2^{\log^* N} \log n)))</td>
</tr>
</tbody>
</table>

Multiple indexed attributes

- Naïve way is to duplicate the entire stack of states of \(U\) and \(S\) and use one per attribute
- We can do better: ORAM state (the largest part) is shared, index and \(DS\) per attribute
- Need to split privacy budget \(\epsilon\) among attributes (Composition Theorem)
Storage efficiency is defined as the sum of the bit-lengths of the records in a database relative to the bit-length of a corresponding encrypted database. Communication efficiency is defined as the sum of the lengths of the records in bits whose search keys satisfy the query relative to the actual number of bits sent back as the result of a query. Efficiency is defined as $(a_1, a_2)$, where $a_1$ is a multiplicative term and $a_2$ is the additive one.

Epsolute’s storage efficiency is $(\mathcal{O}(1), 0)$. Communication efficiencies for different query and DP types are shown.

<table>
<thead>
<tr>
<th></th>
<th>Point Query</th>
<th>Range Query</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pure DP</td>
<td>$(\mathcal{O}(\log n), \mathcal{O}(\log N \log n))$</td>
<td>$(\mathcal{O}(\log n), \mathcal{O}(\log N \log n))$</td>
</tr>
<tr>
<td>Approximate DP</td>
<td>$(\mathcal{O}(\log n), \mathcal{O}(\log n))$</td>
<td>$(\mathcal{O}(\log n), \mathcal{O}(2^{\log^* N} \log n))$</td>
</tr>
</tbody>
</table>

Multiple indexed attributes

- Naïve way is to duplicate the entire stack of states of $U$ and $S$ and use one per attribute
- We can do better: ORAM state (the largest part) is shared, index and $DS$ per attribute
- Need to split privacy budget $\epsilon$ among attributes (Composition Theorem)
Parallel $\mathcal{E}$psolute: the choice of separate vs shared $\mathcal{DS}$

- Split $\mathcal{U}$ and $\mathcal{S}$ state into $m$ ORAMs, run as separate machines
- Partition records randomly (by ID) into $m$ partitions, generate $m$ inverted indexes

**No-$\gamma$ method: $\mathcal{DS}$ per ORAM**
- Composition of disjoint datasets: take max $\epsilon$
- Each ORAM replies with $\left(1 + \gamma\right)\frac{k_0}{m}$ records, where $k_0$ is a required number of records
- To bound probability to $\beta$, use $\gamma = \sqrt{-3m \log \beta k_0}$

**$\gamma$-method: shared $\mathcal{DS}$**
- Same number of records per ORAM
- Use $\gamma$ as in no-$\gamma$ method, except
  - $k_0 \leftarrow k_0 + \frac{\log N}{\epsilon}$ for point queries
  - $k_0 \leftarrow k_0 + \frac{\log^{1.5} N}{\epsilon}$ for range queries

<table>
<thead>
<tr>
<th>Point Query</th>
<th>Range Query</th>
</tr>
</thead>
<tbody>
<tr>
<td>no-$\gamma$-method</td>
<td>$\mathcal{O}\left(\left(1 + \sqrt{-3m \log \beta k_0 / m}\right) \log \frac{n}{m} \log n\right)$, $\mathcal{O}\left(\frac{\log N}{m \log n}\right)$</td>
</tr>
<tr>
<td>$\gamma$-method</td>
<td>$\mathcal{O}\left(\left(1 + \sqrt{-3m \log \beta / k_0}\right) \log \frac{n}{m} \left(1 + \frac{\log N}{\epsilon}\right)\right)$, 0</td>
</tr>
</tbody>
</table>

Communication efficiencies for different query types and methods.
Parallel $\mathcal{E}$psolute: the choice of separate vs shared $\mathcal{DS}$

- Split $\mathcal{U}$ and $\mathcal{S}$ state into $m$ ORAMs, run as separate machines
- Partition records randomly (by ID) into $m$ partitions, generate $m$ inverted indexes

No-$\gamma$ method: $\mathcal{DS}$ per ORAM
- Composition of disjoint datasets: take max $\epsilon$
- Each ORAM replies with $(1 + \gamma)^{\frac{k_0}{m}}$ records, where $k_0$ is a required number of records
- To bound probability to $\beta$, use $\gamma = \sqrt{-\frac{3m\log\beta}{k_0}}$

$\gamma$-method: shared $\mathcal{DS}$
- Same number of records per ORAM
- Use $\gamma$ as in no-$\gamma$ method, except
  - $k_0 \leftarrow k_0 + \frac{\log N}{\epsilon}$ for point queries
  - $k_0 \leftarrow k_0 + \frac{\log 1.5 N}{\epsilon}$ for range queries

<table>
<thead>
<tr>
<th>Query Type</th>
<th>no-$\gamma$-method</th>
<th>$\gamma$-method</th>
</tr>
</thead>
<tbody>
<tr>
<td>Point Query</td>
<td>$O\left(\left(1 + \sqrt{-\frac{3m\log\beta}{k_0}}\right) \log \frac{n}{m}, O\left(\frac{\log N}{\epsilon} \cdot m \log n\right)\right)$</td>
<td>$O\left(\left(1 + \sqrt{-\frac{3m\log\beta}{k_0}}\right) \log \frac{n}{m}, O\left(\frac{\log 1.5 N}{\epsilon} \cdot m \log n\right)\right)$</td>
</tr>
<tr>
<td>Range Query</td>
<td>$O\left(\left(1 + \sqrt{-\frac{3m\log\beta}{k_0}}\right) \log \frac{n}{m} \left(1 + \frac{\log N}{\epsilon}\right), 0\right)$</td>
<td>$O\left(\left(1 + \sqrt{-\frac{3m\log\beta}{k_0}}\right) \log \frac{n}{m} \left(1 + \frac{\log 1.5 N}{\epsilon}\right), 0\right)$</td>
</tr>
</tbody>
</table>

Communication efficiencies for different query types and methods.
Parallel Epsolute: the choice of separate vs shared DS

- Split $U$ and $S$ state into $m$ ORAMs, run as separate machines
- Partition records randomly (by ID) into $m$ partitions, generate $m$ inverted indexes

**No-$\gamma$ method: DS per ORAM**
- Composition of disjoint datasets: take max $\epsilon$
- Each ORAM replies with $(1 + \gamma)\frac{k_0}{m}$ records, where $k_0$ is a required number of records
- To bound probability to $\beta$, use $\gamma = \sqrt{-\frac{3m \log \beta}{k_0}}$

**$\gamma$-method: shared DS**
- Same number of records per ORAM
- Use $\gamma$ as in no-$\gamma$ method, except
  - $k_0 \leftarrow k_0 + \frac{\log N}{\epsilon}$ for point queries
  - $k_0 \leftarrow k_0 + \frac{\log^{1.5} N}{\epsilon}$ for range queries

<table>
<thead>
<tr>
<th>Point Query</th>
<th>Range Query</th>
</tr>
</thead>
<tbody>
<tr>
<td>no-$\gamma$-method</td>
<td>$O\left((1 + \sqrt{-\frac{3m \log \beta}{k_0}}) \log \frac{n}{m}, O\left(\frac{\log N}{\epsilon} m \log n\right)\right)$, $O\left(1 + \sqrt{-\frac{3m \log \beta}{k_0}} \log \frac{n}{m} (1 + \frac{\log N}{\epsilon})\right)$</td>
</tr>
<tr>
<td>$\gamma$-method</td>
<td>$O\left((1 + \sqrt{-\frac{3m \log \beta}{k_0}}) \log \frac{n}{m} (1 + \frac{\log N}{\epsilon})\right)$, 0</td>
</tr>
</tbody>
</table>

Communication efficiencies for different query types and methods.
Parallel $\mathcal{E}$psolute: the choice of separate vs shared $\mathcal{DS}$

- Split $\mathcal{U}$ and $\mathcal{S}$ state into $m$ ORAMs, run as separate machines
- Partition records randomly (by ID) into $m$ partitions, generate $m$ inverted indexes

**No-$\gamma$ method: $\mathcal{DS}$ per ORAM**

- Composition of disjoint datasets: take max $\epsilon$
- Each ORAM replies with $(1 + \gamma)\frac{k_0}{m}$ records, where $k_0$ is a required number of records
- To bound probability to $\beta$, use $\gamma = \sqrt{-3m \log \beta \frac{k_0}{k_0}}$

**$\gamma$-method: shared $\mathcal{DS}$**

- Same number of records per ORAM
- Use $\gamma$ as in no-$\gamma$ method, except
  - $k_0 \leftarrow k_0 + \frac{\log N}{\epsilon}$ for point queries
  - $k_0 \leftarrow k_0 + \frac{\log^{1.5} N}{\epsilon}$ for range queries

<table>
<thead>
<tr>
<th>Point Query</th>
<th>Range Query</th>
</tr>
</thead>
<tbody>
<tr>
<td>no-$\gamma$-method</td>
<td>$O\left(\left(1 + \sqrt{-3m \log \beta \frac{k_0}{k_0}}\right) \log \frac{n}{m} \cdot O\left(\frac{\log N}{\epsilon} m \log n\right)\right)$</td>
</tr>
<tr>
<td>$\gamma$-method</td>
<td>$O\left(\left(1 + \sqrt{-3m \log \beta \frac{k_0}{k_0}}\right) \log \frac{n}{m} \left(1 + \frac{\log N}{\epsilon}\right)\right)\cdot 0$</td>
</tr>
</tbody>
</table>

Communication efficiencies for different query types and methods.
Parallel $\mathcal{E}$psolute diagram (with improvements)

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

Client

Untrusted server party $S$

Trusted user party $U$

Lightweight ORAM machine

KVS Store

DP histogram

DP tree

B+ tree

Application

User

KVS Store

KVS Store

KVS Store
EXPERIMENTS
Experiments setup

- Implemented in C++, PathORAM and B+ tree are modules, OpenSSL for crypto
- Run on GCP in different regions, 8 storage VMs, 8 ORAM VMs, one client
- Two real datasets (salaries) and one synthetic (uniform) dataset of sizes 100K, 1M and 10M
- Default setting
  - $\epsilon = \ln 2$ and $\beta = 2^{-20}$
  - 1M uniformly sampled 4 KiB records
  - selectivity 0.5%
  - $\gamma$-method
- Mechanisms (besides Epsolute)
  - MySQL and PostgreSQL
  - Linear Scan (download everything every query)
  - Shrinkwrap [28] (adapted range queries from their source code)
How practical is our system compared to the most efficient and most private real-world solutions?

Three orders of magnitude faster than Shrinkwrap [28], 18 times faster than the linear scan and only 4–8 times slower than a conventional database.
### Question 2: storage

**How practical is the storage overhead?**

<table>
<thead>
<tr>
<th>$n$</th>
<th>$1$ KiB</th>
<th>$4$ KiB</th>
<th>$16$ KiB</th>
</tr>
</thead>
<tbody>
<tr>
<td>$10^5$</td>
<td>400 KiB</td>
<td>400 B</td>
<td>1.6 MB</td>
</tr>
<tr>
<td></td>
<td>396 MB</td>
<td>4.6 MB</td>
<td>51 MB</td>
</tr>
<tr>
<td>$10^6$</td>
<td>3.9 MB</td>
<td>400 B</td>
<td>3.9 MB</td>
</tr>
<tr>
<td></td>
<td>3.2 GB</td>
<td>15 MB</td>
<td>1.6 MB</td>
</tr>
<tr>
<td>$10^7$</td>
<td>40 MB</td>
<td>400 B</td>
<td>40 MB</td>
</tr>
<tr>
<td></td>
<td>24 GB</td>
<td>99 MB</td>
<td>146 MB</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$N$</th>
<th>$100$</th>
<th>$10^4$</th>
<th>$10^6$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$1.6$ MB</td>
<td>$414$ GB</td>
<td>$514$ GB</td>
<td></td>
</tr>
</tbody>
</table>

*Italic* values are estimated. $S$ to $U$ storage size ratio is 85, 414 and over 2 000.
**Question 3: varying parameters**

How different inputs and parameters of the system affect its performance?

$\epsilon$ strictly contributes to the amount of noise, which grows exponentially as $\epsilon$ decreases.

Overhead expectedly grows with the result size.
Question 4: scalability

How well does the system scale?

The $\gamma$-method provides substantially better performance and storage efficiency, and when using this method the system scales \textbf{linearly} with the number of ORAMs.
What is the impact of supporting multiple attributes?

The overhead increases only slightly due to a lower privacy budget.
Epsolute: Efficiently Querying Databases While Providing Differential Privacy

Differential Privacy, ORAM, differential obliviousness, sanitizers

[42] DOI: 10.1145/3460120.3484786

Dmytro Bogatov, Georgios Kellaris, George Kollios, Kobbi Nissim, Adam O’Neill
dmytro@bu.edu, kellaris@bu.edu, gkollios@cs.bu.edu,
kobbi.nissim@georgetown.edu, adamo@cs.umass.edu

Built from 078878e4 on October 17, 2021
References


References


References


