The PINED-RQ Family: Differentially Private Indexes for Range Query Processing in Clouds

Tristan Allard (Univ Rennes, CNRS, Irisa)

Joint work with R. Akbarinia (INRIA, LIRMM), A. El Abbadi (UCSB), L. d’Orazio (Univ Rennes, CNRS, IRISA), E. Pacitti (Univ Montpellier, LIRMM, CNRS), C. Sahin (UCSB), H. V. Tran (Univ Rennes, CNRS, IRISA)

Sosysec Seminar
13th Nov 2020

Thanks to C. Sahin and H. V. Tran for sharing some pictures and slides.
Who Am I?

Main Focus
Design privacy-preserving personal data management and analysis systems and explore the resulting privacy-efficiency-quality tradeoffs

Main Tools

- **Distribution**: client-server architectures with untrusted parties, completely distributed architectures.

- **Sanitization models and mechanisms**: perturbation that satisfies differential privacy or variants

- **Encryption mechanisms**: block ciphers (e.g., symmetric AES), homomorphic encryption (e.g., additively-homomorphic Paillier), etc.

*Leitmotiv: perturbation techniques as building blocks for privacy-preserving algorithms.*
Why Using Differential Privacy as a Building Block?

In general:

- Encryption alone: computation may be costly or may not cope with churn when distributed, the final result may reveal too much.
- Differential privacy can allow to (for example): switch to cleartext while keeping sound protections (perturbation), limit the leaks from the final result of encrypted functions.
- It is interesting! (security models, privacy budget management, algorithms adaptation).

A specific illustration below: the PINED-RQ family [16, 17] (and ongoing reviews).
Progress of the Talk

Introduction

Problem Definition

PINED-RQ

The FRESQUE Extension (Quick Overview)

Conclusion
Substantial advances in outsourced data management techniques but...

Strong security concerns hamper the adoption of cloud solutions for private data

And naive encryption of the complete database is not a viable solution
Secure Range Query?

- **Goal**: Answer to queries that involve numerical comparisons with realistic performances
- **Example query**: SELECT * FROM students WHERE grade ≥ 3 AND grade ≤ 4
- **A basic primitive for various applications (e.g., transactions, analytics)**
Objective

- Let a **honest-but-curious** cloud...
- Answer to **selection range queries** over encrypted personal data...
- While providing **differentially private guarantees**...
- Together with **realistic performances**.

![Diagram of data provider and consumers interacting with a cloud](image)
Related Work in a Nutshell

- Approaches based on bucketization do not provide formal privacy guarantees (e.g., [10, 12, 11])
- Approaches based on order-preserving encryption schemes are vulnerable to statistical attacks (e.g., [3, 4])
- Approaches based on symmetric searchable encryption suffer from high space and/or times requirements (e.g., [13, 7])
Approach:

- **Data provider**: compute a one-dimensional differentially-private index inspired from B+-Trees over the records encrypted by any usual secret key semantically-secure encryption scheme (e.g., AES), and send both to the cloud.

- **Cloud**: receive range queries (cleartext) and answer them by returning encrypted records based on the differentially-private (cleartext) index.

- **Both**: support updates! (Inserts, modifies, and deletes.)
Introduction

Problem Definition

PINED-RQ

The FRESQUE Extension (Quick Overview)

Conclusion
Basic Data Structures

▶ **Dataset (private)**: a relation $\mathcal{D}(A_1, \ldots, A_d)$. Considered to be private.

▶ **Queries (public)**: non-aggregate single-dimensional range queries over a single attribute $A_q$.

▶ **Exact results (or relevant records)**: set of records $\mathcal{R}$ in $\mathcal{D}$ satisfying the query $\mathcal{Q}$.

▶ **Outsourced data structures (satisfy differential privacy)**:
  
  ▶ Encrypted version of the dataset $\overline{\mathcal{D}}$ (semantically secure encryption scheme)
  
  ▶ Index $\mathcal{I}(A_q)$ over the queriable attribute of $\mathcal{D}$, pointing to the encrypted records $\overline{r} \in \overline{\mathcal{D}}$. 
Inherent loss of information due to the differentially private perturbation

Quantification of the quality we reach by recall and precision measures

Definition (Recall and Precision)

Given a query $Q$, with an exact set of relevant records $R$ in $D$, while the set of records returned by the cloud is $\tilde{R}$, then the recall $r$ and precision $p$ of $\tilde{R}$ are: $r = |R \cap \tilde{R}|/|R|$ and $p = |R \cap \tilde{R}|/|\tilde{R}|$. 
Original Differential Privacy Model

\( \epsilon \)-differential privacy (from [8])

A random function \( f \) satisfies \( \epsilon \)-differential privacy iff: For all \( D \) and \( D' \) differing in at most one record, and for any possible output \( S \) of \( f \), then it is true that:

\[
Pr[f(D) = S] \leq e^{\epsilon} \times Pr[f(D') = S]
\]

- \( f \): an aggregate query perturbed (originally)
- “For all \( D \) and \( D' \)” : all possible datasets
- “\( D \) and \( D' \) differing in at most one record” : \( D \) is \( D' \) with one tuple more or one tuple less
- \( \epsilon \) : the privacy parameter, public, common values: 0.01, 0.1, \( \ln 2 \), \( \ln 3 \)
Laplace Mechanism

Given $\epsilon$, adding to the output of a COUNT aggregate query a random variable sampled from a Laplace distribution with mean 0 and scale factor $1/\epsilon$ satisfies $\epsilon$-differential privacy [9].

**Figure:** Laplace $(0, 1/0.01)$
Nice Properties

- **Self-composability**: composing the outputs of two independent releases sanitized by differentially-private function(s) satisfies differential privacy:
  - Where $\epsilon_{\text{final}} = \sum \epsilon_i$ if input datasets are **not** disjoint
  - Or $\epsilon_{\text{final}} = \max \epsilon_i$ otherwise

- **No breach from post-processing**: Any function applied to a differentially-private input produces a differentially-private output
Computational Differential Privacy

- Original differential privacy provides information theoretic guarantees...
- But when combined with a semantically secure encryption scheme, the end-to-end guarantees become computational!

**Definition ($\epsilon_n$-SIM-CDP privacy [14])**

Randomized function $f_n$ provides $\epsilon_n$-SIM-CDP if there exists a function $F_n$ that satisfies $\epsilon_n$-differential-privacy and a polynomial $p(\cdot)$, such that for every input dataset $D$, every probabilistic polynomial time adversary $A$, every auxiliary background knowledge $\zeta \in \{0, 1\}^*$, and every sufficiently large $n \in \mathbb{N}$, it holds that:

$$|\Pr[A_n(f_n(D, \zeta)) = 1] - \Pr[A_n(F_n(D, \zeta)) = 1]| \leq \frac{1}{p(n)}$$
A probabilistic relaxation of $\epsilon_n$-SIM-CDP:

**Definition** ($(\epsilon, \delta)_n$-Probabilistic-SIM-CDP)

A randomized function $f_n$ is said to provide $(\epsilon, \delta)_n$-Probabilistic-SIM-CDP, if it provides $\epsilon_n$-SIM-CDP [14] to each individual with probability greater than or equal to $\delta$, where $\delta \in ]0, 1]$.
Problem(s) I

PINED-RQ

Design the following functions while satisfying $(\epsilon, \delta)_n$-Probabilistic-SIM-CDP and achieving realistic performance and quality levels:

1. **Index creation (CREATE):** create the differentially private data structures

2. **Query execution:** answer to range queries based on the index

3. **Index updates (INSERT, MOD/DEL):** maintain the index against inserts, modifies, and deletes operations (*will not be presented here for time reasons*).
Problem(s) II

Extensions

▶ **Rationalize the software architecture (FRESQUE):** streamline and parallelize the CREATION and INSERT functions.
  ⇒ Support high ingestion rates (100k+ records per second in our experiments).

▶ **Diversify the kind of index (PARADOT):** index the primary key attribute (based on PINED-RQ augmented with bitmaps).
  ⇒ Support an infinite number of updates.
Progress of the Talk

Introduction

Problem Definition

PINED-RQ

The FRESQUE Extension (Quick Overview)

Conclusion
Step 1: Compute $\mathcal{I}(A_q)$ (B+-Trees + DP histograms)

Create the index $\mathcal{I}(A_q)$

- $\mathcal{I}(A_q)$ is a balanced tree with a fixed branching factor (see, e.g., [15] for setting it)
- Each node is a bin, each level represents a histogram over $A_q$
- Leaf nodes are *unit-size* bins and point to encrypted records
- Perturb each bin based on the usual Laplace mechanism and apply known consistency constraints and privacy budget distribution strategies [6, 15]

**Figure: Step 1 before perturbation**
Step 2: Compute $\overline{D}$

Create the encrypted dataset $\overline{D}$

- **Main goal**: consistency between perturbed leaf nodes and number of encrypted records
- **Positive noise**: add dummy records
- **Negative noise**: remove records, put them in the overflow array of the leaf
- **Size of overflow arrays (fixed)**: computed based on Laplace CDF $s.t.$ noise is lower with a given probability (e.g., 99.99%)
Step 2: Compute $\overline{D}$ II

![Diagram showing the process of computing $\overline{D}$ II after encryption]

**Figure**: Step 2 after encryption
Privacy Guarantees of CREATE

**Theorem**

The CREATE function, in charge of computing $\mathcal{I}(A_q)$, $\mathcal{D}$, and the overflow arrays satisfies $(\epsilon, \delta)_n$-Probabilistic-SIM-CDP as defined in Definition 3.

Proofs in the paper.
Query Processing Strategy

Naive Query Processing

Given a range query $Q$:

1. Start at the root of the tree
2. Traverse the child of any node that has a non-negative intersection with $Q$
3. If a leaf node has a positive count $Q$ : return the records pointed to by the corresponding node.
4. Otherwise : always return the overflow array (high recall priority).
5. (On data consumers) filter out false positives

*Experimental results actually show that this strategy is sufficient!*
Settings

- **Environment**: Java implementation, Windows 7 OS, i5-2320 3 GHz CPU, 8 GB RAM.

- **Parameters**: Branching factor is set to 16, total privacy budget $\epsilon_{total} = 1$, domain of $A_q$ normalized to $[0, 100]$, size of overflow arrays with 99.99% confidence interval.

- **Synthetic datasets**: uniform or Zipfian with a skewness of 1, with 0.5 million records.

- **Real datasets**: Gowalla [5] (locations and times, 6,442,890 records, relatively uniform), and US Postal Employees [1], USPS, dataset (394,763 records after cleaning, highly skewed).

- **Queries**: sample 1000 random queries inside each of the following ranges: 1%, 5%, 10%, 25%, 50%, and 75% of the entire domain.
Quality (sample) I

Recall Rate (%)

Range Size (% of domain)

Figure: Recall (varying range size)
Quality (sample) II

Figure: Precision (varying range size)
Quality (sample) III

**Figure:** Precision (varying range size) - workload follows data distribution (own) or not (uniform)
Index Scan Timing

Figure: Index scan time (Gowalla, 500k records)
Progress of the Talk

Introduction

Problem Definition

PINED-RQ

The FRESQUE Extension (Quick Overview)

Conclusion
Weaknesses of PINED-RQ

PINED-RQ generates bottlenecks

- **Batch publishing**
  ⇒ FRESQUE streams the incoming records to the cloud.

- **Naive data structures**
  ⇒ FRESQUE uses arrays extensively (constant-time accesses)

- **Naive architecture**
  ⇒ FRESQUE clearly defines the software modules of PINED-RQ and parallelizes them on a shared-nothing infrastructure.
Focus on the Architecture of FRESQUE

Figure: The Architecture of FRESQUE
Experimental Environment

- FRESQUE implemented in Java 1.8.0.
- Galactica platform, cluster of 17 nodes (Ubuntu 14.04.4 LTS)
- Incoming data rate: 200k records per second.
- PINED-RQ : fanout is 16, default $\epsilon$ is 1, $\delta$ is set to 99%.
- Results: average of ten experiments.

Table: Experimental environment

<table>
<thead>
<tr>
<th>Component</th>
<th>CPU (2.4 GHz)</th>
<th>Memory (GB)</th>
<th>Disk (GB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dispatcher</td>
<td>4</td>
<td>8</td>
<td>80</td>
</tr>
<tr>
<td>Merger</td>
<td>4</td>
<td>8</td>
<td>80</td>
</tr>
<tr>
<td>Checking node</td>
<td>4</td>
<td>8</td>
<td>80</td>
</tr>
<tr>
<td>Computing node</td>
<td>2</td>
<td>2</td>
<td>20</td>
</tr>
<tr>
<td>Data source</td>
<td>4</td>
<td>16</td>
<td>80</td>
</tr>
<tr>
<td>Cloud</td>
<td>16</td>
<td>64</td>
<td>160</td>
</tr>
</tbody>
</table>
Ingestion Throughput

Figure: Ingestion throughput of FRESQUE
Ingestion Throughput - Comparison I

Figure: Ingestion throughput of FRESQUE - Improvement over our non-parallel version of PINED-RQ++ (≈ similar to PINED-RQ)
Ingestion Throughput - Comparison II

Figure: Ingestion throughput degradation of FRESQUE - Comparison with our previous versions
Progress of the Talk

Introduction

Problem Definition

PINED-RQ

The FRESQUE Extension (Quick Overview)

Conclusion
Key Take-Aways

▶ The PINED-RQ family: differential privacy together with encryption for efficient encrypted data querying
▶ Secure against a honest-but-curious cloud, where:
  ▶ Security model is the simulation-based flavor of computational differential privacy [14]
  ▶ Differential privacy is satisfied probabilistically for PINED-RQ and FRESQUE
  ▶ Cleartext queries are assumed to be innocuous
▶ Overall, experimental evaluations show:
  ▶ High recall and precision levels. Results depend on the size of the range (the larger the better), and on data distribution (high impact of the noise on the parts of the domain that contain few data)
  ▶ Index scan time is realistic (much faster than related works).
  ▶ High ingestion throughput for FRESQUE (orders of magnitude higher than related works).
Thank you !
Data Universe, Asbury Park Press.


Order-preserving encryption for numeric data.

Order-preserving encryption revisited: Improved security analysis and alternative solutions.

Differentially private spatial decompositions.

Practical private range search revisited.

[8] C. Dwork.
Differential privacy.
Springer-Verlag.

Calibrating noise to sensitivity in private data analysis.


Understanding hierarchical methods for differentially private histograms.

A differentially private index for range query processing in clouds.
In *ICDE ’18*, 2018.

Range query processing for monitoring applications over untrustworthy clouds.
In *EDBT ’19*, 2019.