When the Signal is in the Noise: Exploiting Diffix's Sticky Noise

Andrea Gadotti*, Florimond Houssiau*, Luc Rocher*, Benjamin Livshits, Yves-Alexandre de Montjoye
Anonymization
A different model: data query systems

From de-identification...

- Individual-level data
- No control over analyses

... to data query systems

- Aggregation
- Additional security and privacy measures

WHAT IF ANALYST IS MALICIOUS?
"How many people named Bob have a salary ≤ £2000"
Q1 = "How many people have a salary ≤ £2000"

Q2 = "How many people not named Bob have a salary ≤ £2000"

\[ Q1 - Q2 = [0 \text{ or } 1] \]

This is called differencing attack.
Random noise to prevent privacy attacks

“How many people have a salary ≤ £1500?”
Reconstruction attacks and differential privacy

First reconstruction attack (Dinur and Nissim, 2003). If noise is not enough → attacker can reconstruct the full dataset in polynomial time.

Since then, the attack has been generalized and improved.

One solution: differential privacy (Dwork et al., 2006).

Pros:
• provable and meaningful guarantee
• mathematical framework for privacy/utility

Cons (as of today):
• adds too much noise in many cases
• hard to allow many queries
• hard to provide good usability/flexibility
A heuristic-based data query system: Diffix
Diffix is a privacy-preserving database system

Diffix is a patented commercial system developed by the company Aircloak and researchers at the Max Planck Institute for Software Systems.

Diffix operates as an **SQL proxy** between the analyst and the database.

- **Rich SQL syntax**
- **Little noise**
- **Infinite queries**
Diffix’s noise mechanism: sticky noise

An analyst submits a (count) SQL query $Q$ to Diffix:

```
SELECT count(*)
FROM table
WHERE condition_1 AND condition_2 [AND ...]
```

To which Diffix responds with:

$output = \text{true count} + \text{static noise} + \text{dynamic noise}$

$\text{static noise} \leftarrow \text{query syntax of } Q$

$\text{dynamic noise} \leftarrow \text{query syntax and user set of } Q$

Both noises are \textit{sticky}: issuing the same query gives the same noise.
Diffix’s noise mechanism: sticky noise

\[ Q = \text{count}(\ \text{age} = 40 \land \text{dept} = \text{Computing} \land \text{high-salary} = \text{True}) \]

Each noise $\sim \mathcal{N}(0,1)$

More measures...
Our noise-exploitation attack(s) on Diffix:
Exploiting data-dependent noise
Attack model and assumptions

- Dataset has \( d \) attributes
  \[ \{a_1, \ldots, a_{d-1}, s\} \]
- One target at a time: Bob
- Attacker wants to infer Bob’s attribute \( s \) (binary).
- Attacker knows:
  - Bob’s record is in the dataset
  - The value of \( k \) attributes about Bob

**Example** (with \( d=3, k=2 \))

Dataset attributes:
\[ \{\text{age, department, high-salary}\} \]

Secret attribute: high-salary

Bob’s record: (40, Computing, true)

Attacker knows: (40, Computing)
Differential attack

Q1 = count(dept = Computing \land high-salary = True)
Q2 = count(age \neq 40 \land dept = Computing \land high-salary = True)

Bob:
age = 40
department = computing
high-salary = ? (unique)

Output of Q1 – Q2

True count 1
- True count 2

A. Gadotti & L. Rocher / When the Signal is in the Noise
Differential attack

Q1 = \text{count} ( \text{dept} = \text{Computing} \land \text{high-salary} = \text{True} )

Q2 = \text{count} ( \text{age} \neq 40 \land \text{dept} = \text{Computing} \land \text{high-salary} = \text{True} )

Output of Q1 - Q2 if high-salary = True

Bob:
\begin{align*}
\text{age} &= 40 \\
\text{dept} &= \text{computing} \\
\text{high-salary} &= \text{? (unique)}
\end{align*}
Differential attack

Output of Q1 – Q2 if high-salary = False

\[
Q1 = \text{count}( \text{dept} = \text{Computing} \land \text{high-salary} = \text{True} )
\]

\[
Q2 = \text{count}( \text{age} \neq 40 \land \text{dept} = \text{Computing} \land \text{high-salary} = \text{True} )
\]

Bob:
age = 40
department = computing
high-salary = ?
(unique)
Differential attack

if high-salary = True

\[ Q_1 - Q_2 \sim N(\mu=0, \sigma=2) \]

if high-salary = False

\[ Q_1 - Q_2 \sim N(\mu=1, \sigma=2k+2) \]
The cloning attack

Main issues with the differential attack:

1. Assumes that Bob is unique
2. Attack queries likely to be suppressed

Improved attack: cloning attack
- Much better accuracy
- Relies on weaker notion of uniqueness

Accuracy not great in some cases
Value-uniqueness

**Definition:** A record is value-unique w.r.t. a set of attributes \(\{a_1, \ldots, a_k\}\) if all records sharing the same attributes also have the same secret attribute.

**Example**

<table>
<thead>
<tr>
<th>age</th>
<th>dept</th>
<th>high-salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>40</td>
<td>computing</td>
<td>true</td>
</tr>
<tr>
<td>40</td>
<td>computing</td>
<td>true</td>
</tr>
<tr>
<td>34</td>
<td>math</td>
<td>false</td>
</tr>
<tr>
<td>34</td>
<td>math</td>
<td>true</td>
</tr>
</tbody>
</table>

**Note:** Value-uniqueness is detected automatically by the cloning attack.
Results for the cloning attack

- Attacked and correctly inferred: \(~90\%\) of all users
- Modified attack: 32 queries/user
Aircloak’s proposed patch

Aircloak’s patch

Remove “dangerous” (low effect) conditions from queries (depending on data).

Comment. Does not address core vulnerability and potentially introduces new one.

Expected patch date: Q4 2019
Other attacks on Diffix

**Membership inference attack**  
*by A. Pyrgelis, C. Troncoso, E. De Cristofaro*

**idea:** infer whether an individual is in the dataset by training a classifier to tell this from aggregate data.


**Linear reconstruction attack**  
*by A. Cohen, K. Nissim*

**idea:** send queries targeting “random enough” sets of users and use the results to build a linear system, then reconstruct the database from it.

Conclusions

- Anonymization 
- Data query systems
- Relying on single mechanism is risky
- Defense-in-depth (e.g. query auditing, query rate limiting, etc.)

but also...

- alternatives to differential privacy are useful
- transparency is fundamental
Thank you for your attention!

Find out more at: https://cpg.doc.ic.ac.uk/blog/aircloak-diffix-signal-is-in-the-noise/
Backup
What does it mean?
QBS are a step in the right direction...

Anonymization does not work anymore: “We have currently no reason to believe that an efficient enough, yet general, anonymization method will ever exist for high-dimensional data, as all the evidence so far points to the contrary.”

de Montjoye et al. (2016)

Query-based and question-and-answer systems are clearly the way forward and Diffix is one of the first query-based systems developed and brought to the industry.

... but require a security approach

Fixes to our attacks exist: Aircloak announced that they have patched their system (removing low-impact conditions). However, this also leaks information which could open-up new avenue for attacks.

Moving forward, a security mindset:

- No system is perfect, even Differentially Private ones.
- Layered approach incl. anomaly detection
- Need for constant attack and defense
Defining Privacy for Query-Based Systems

Differential Privacy
The main definition of privacy used in QBS in the literature is **differential privacy**. It essentially requires that the output of an algorithm should not depend on the presence of one individual in the dataset. Formally, \( M \) is epsilon-DP if, for any two neighboring datasets \( D \) and \( D' \) and output \( y \):

\[
\Pr[M(D) = y] \leq e^\varepsilon \Pr[M(D') = y]
\]

Very strong and powerful definition:
- Resilient to many attacks
- *provably* optimal*

Practical limitations
However, it is often **hard to implement in practice**, in particular for general-purpose analytics such as SQL databases.

Differential Privacy (DP) is really appropriate for some problems (Rappor, Apple’s DP, PINQ, Chorus), but hard to generalize to many real-world problems.

Main issues:
- Usability
- Limits on the number of queries.
A solution: security mechanisms and QBS systems

Instead of trying to alter the dataset “once and for all” then lose access over it, we could retain the data on a server and answer questions about it (query-based system). This way, the data is never published.

→ Keep control over what we disclose.

→ Computer security principles apply (authorization, encryption, secure access, … )
There are many attacks on query-based systems

→ An adversary can still ask sensitive questions:
  “How many people named Florimond have a salary ≤ £1500”

→ A simple solution could be to add random noise to the answer of queries...

⇒ **Averaging attacks**: repeat the same query several times to average out the noise.

→ You can use caches, but needs to be carefully designed (more on this later).

→ Another solution is to refuse to answer queries that concern less than $k$ users (Query-set size restriction).

Is this secure?

⇒ **Difference attacks**: combine queries to obtain the result you want. Example:
Q1 = “How many people have a salary ≤ £1500”
Q2 = “How many people not named Florimond have a salary ≤ £1500”
→ Q1 - Q2 = answer!
Assumptions (and notations)

We make these assumptions:

1. The attacker knows that their target, Bob, is in the dataset (Bob’s record is denoted by $x$);
2. The attacker has some auxiliary information about Bob, as a set of Bob’s attributes, $A : x^{(A)}$;
3. The attacker wants to infer a binary secret attribute $s$ of Bob’s record, $x^{(s)}$;
4. Bob is unique according to the attributes $A$.*

**known attributes:**

$A = (\text{age, gender, department})$

$x^{(A)} = (40, M, \text{computing})$

**secret attribute:**

$s = \text{promoted}$

$x^{(s)} = \text{False}$
The noise depends on the data

Our attacks exploit the noise structure, and in particular the fact that the noise depends on the data.

Idea:
- \( Q_1 \): includes Bob if \( x^{(s)} = True \)
- \( Q_2 \): \( Q_1 \) + a condition that always excludes Bob
- Most noises cancel out in \( Q_1 - Q_2 \) if \( x^{(s)} = True \), as the query-sets are identical

We developed two attacks:

1. The differential attack is our initial attack, which exploits this process;
2. The cloning attack improves on this attack by using dummy conditions to improve statistical power.
Our attack succeeds on 64-69% of unique users

We measure:
- The fraction of users that are unique according to $k^*$ known attributes.
- The fraction of these users for which our attack succeeds (correctly inferred)
The cloning attack

This attack improves on the differential attack:

- It is less sensitive to bucket suppression;
- It is statistically more powerful;
- It “self-validates” its assumptions.

The idea behind this attack is to use *dummy conditions* which do not affect the output of the query (i.e. that are always true for this query).

*For instance:* \( \text{count}( \text{age} = 42 \land \text{age} \leq 50 ) \)
Assumption self-validation: subset exploration

Assumptions

In this attack, the attacker needs a subset $A'$ and an attribute $u$, $(A', u)$: $A' \subseteq A$, $u \in A \setminus A'$. Two assumptions must be met on $(A', u)$:

1. The user is value-unique according to $(A', u)$.
2. The queries of the attack are not bucket suppressed.

Subset exploration

- The attacker explores subsets of $A$ until it finds a “good” pair $(A', u)$: $\rightarrow \{\text{age}\}, \{\text{age, dept}\}, \{\text{dept, sex}\}, \ldots$

- We proposed two tests to estimate whether the assumptions are met:

1) Is $\text{count}(A' = x^{(A')}) \land u = x^{(u)}$ bucket suppressed? (it should)
2) Are the queries of the attack bucket suppressed (they shouldn’t)
Fixes to our attack

As such, we did not find any “easy” fix to our attack that wouldn’t open Diffix to other vulnerabilities.

➔ One could want to remove dummy conditions or low-impact conditions (dept ≠ comp);

➔ But this could in turn be exploited by another attack (...)

We do believe, however, that practical secure design principles could help mitigate the issues, as for any query-based system:

1) Intrusion detection
2) Auditability
3) Increased friction
4) Limited expressiveness
5) ...