Differential Privacy Under Fire

(A. Haeberlen, B.C. Pierce, & A. Narayan)

Neil Pruthi and Justin Sherman

Problem and Motivation

Differential privacy has opened ways to achieve robust, provable privacy guarantees.

But these systems are not immune to covert channel attacks, where information is transferred in a secretive, unauthorized, or illicit manner.

In other words, the differential privacy guarantees may not be enough to protect customer information in practice.

Problem and Motivation (cont.)

Example:

- Netflix API to see how many shows/movies a person watched in a given genre
- An adversary runs a query that always returns zero
- If John Doe has watched adult movies, it takes one hour to complete request
- If John Doe has *not*, it answers the request immediately
- Thus: Differential privacy is upheld on the algorithm's side, but the attacker can learn with perfect certainty whether John Doe has watched adult films
- In other words: diff. privacy is blatantly violated

Attack Model

An attacker doesn't necessarily have to be submitting queries locally.

In that case, with information only sent over the network, the attacker cannot rely on the likes of electromagnetic radiation and power consumption to breach privacy.

This narrows the risk down to 2 indicators (since answer is already ε-diff. private):

- Query completion time
- Privacy budget - i.e., did the system execute query or refuse (if budget exceeded)

Example 1: Timing Attack

```
noisy sum, foreach r in db, of {
    if embarrassing(r)
        then { pause for 1 second };
    return 0
}
```

Adversary purposely sends bad value → figures out how quick the exit / default answer is vs. the normal query response → then can figure out exactly how many rows have embarrassing values

Example 2: State Attack

```
found = false;
noisy sum, foreach r in db, of {
    if (found) then { return 1 }
    if embarrassing(r) then {
        found = true;
        return 1
    } else { return 0 }
}
```

Adversary maintains a global variable *between* microqueries → result of each microquery is 0 or 1, depending on whether any previous microquery detected an embarrassing record → and since embarrassing one is generally not last in database, this greatly magnifies the contribute of this one row to result (violating diff. privacy)

Example 3: Privacy Budget Attack

```
noisy sum, foreach r in db, of {
    if embarrassing(r) then {
        run sub-query that uses a lot of the privacy budget
    } else {
        return 0
    }
}
```

Adversary looks for an embarrassing record → invokes some sub-query that will use up some of the remaining privacy budget → then, when outer query is returned, adversary checks how much the privacy budget has decreased (e.g., learning something about the privacy budget consumed by that row)

Approach

The authors test two mechanisms for attack vulnerability:

- Privacy Integrated Queries (PINQ), a LINQ-like API for computing on privacy-sensitive datasets while providing diff. privacy guarantees for underlying records, and
- Airavat, a MapReduce-based system which provides strong security and privacy guarantees for distributed computations on sensitive data.

Both of these are systems that implement a programming language in which every well-typed program is guaranteed to be differentially private. Thus, “(untrusted) non-experts can write as many different algorithms as they like, and the database administrator can rely on the language to ensure that privacy is not being violated.”

Approach (cont.)

Then, they design their own mechanism—Fuzz—to protect against covert channel attacks.

Two main goals:

1. Practicality - it's sufficiently fast and expressive to process realistic queries
2. Effectiveness - it prevents all covert channel attacks possible in the threat model

Testing PINQ

Vulnerable to all of these attacks

The PINQ paper acknowledges the possibility of such attacks, but…

- a rewriter is not provided, and
- the code provided on the PINQ website to sort-of fix this warns that the code “is not hardened or secured” and “should not be used in ‘the wild’”

Testing Airavat

Not vulnerable to privacy budget attacks

- It calculates sensitivity and deducts required amount from privacy budget before query execution begins
- Inherently safe from these attacks

But still vulnerable to timing attacks and state attacks

Building Fuzz

Defending against risk avenue 1: Privacy budget

Fuzz's type checker rules out budget-based channels

Like Airavat, it checks queries before they are executed to determine their privacy cost

And deducts it from the privacy budget before query execution begins

Deduction from privacy budget is independent of database contents!

Building Fuzz (cont.)

Defending against risk avenue 2: Query completion time

- Break each query into “microqueries” that operate on a single row at once
- Enforce that each microquery takes a fixed amount of time
- If time deadline will not be met, abort and return default value

Must ensure nothing else (i.e., memory usage) measurably affects system performance

Pad rows to consume the same amount of memory
Pad all database objects to have the same number of rows
Add appropriate number of dummy rows that consume memory and computation time
Disable garbage collector during microqueries

Evaluation

The paper implements five adversarial queries

Each seeks to vary the query completion time based on whether or not John Doe is in the database

Each query was run a hundred times on each of two versions of the database: one that contains John Doe and another that does not

Results

Query	Attack type	Caml Light runtime (not protected)			Fuzz runtime (protected)		
		Hit	Miss	|Hit−Miss|	Hit	Miss	|Hit-Miss|
weblog-mem	Memory allocation	1.961 s	0.317 s	1.644 s	1.101 s	1.101 s	<1 μs
weblog-gc	Garbage creation	1.567 s	0.318 s	1.249 s	1.101 s	1.101 s	<1 μs
weblog-delay	Artificial delay	1.621 s	0.318 s	1.303 s	1.101 s	1.101 s	<1 μs
weblog-term	Early termination	26.378 s	26.384 s	0.006 s	1.101 s	1.101 s	<1 μs
census-delay	Artificial delay	2.168 s	0.897 s	1.271 s	2.404 s	2.404 s	<1 μs

On the unprotected system:

> "***For four out of the five queries, the completion times for the Hit cases*** **[where John Doe is in the database]** ***were at least one second different*** *from the completion times for the* ***Miss*** *cases,* ***so an adversarial querier could easily have distinguished between the two cases*** *and thus learned with certainty whether or not* [John Doe] *was in the database.*"

Results (cont.)

Query	Attack type	Caml Light runtime (not protected)			Fuzz runtime (protected)		
		Hit	Miss	|Hit−Miss|	Hit	Miss	|Hit-Miss|
weblog-mem	Memory allocation	1.961 s	0.317 s	1.644 s	1.101 s	1.101 s	<1 μs
weblog-gc	Garbage creation	1.567 s	0.318 s	1.249 s	1.101 s	1.101 s	<1 μs
weblog-delay	Artificial delay	1.621 s	0.318 s	1.303 s	1.101 s	1.101 s	<1 μs
weblog-term	Early termination	26.378 s	26.384 s	0.006 s	1.101 s	1.101 s	<1 μs
census-delay	Artificial delay	2.168 s	0.897 s	1.271 s	2.404 s	2.404 s	<1 μs

On Fuzz (the protected system):

The completion times varied by less than a microsecond!

The difference could not even reliably be measured

Strengths / Weaknesses

A weakness: Fuzz is a constant-time solution. In these, "*the size of the database becomes public knowledge, since, except for the most trivial queries, execution time depends on the size of the database.*"

A strength: Fuzz pads all database rows to consume the same amount of memory, and it also pads all database objects to have the same number of rows.

Variable-time solutions, though allowing the size of the database can remain private, present some challenges. "*In order to properly "noise" a resource like time, we must have the ability to both increase and decrease its consumption. While we can clearly increase execution time by adding a delay, we cannot easily decrease it. We can mitigate this problem by adding a default delay T*", and then adding time noise **v**. "*Nevertheless, since noise distributions guaranteeing differential privacy have unbounded support (i.e. P(v) > 0 for all v), there is always a possibility that v < -T, in which can we cannot complete the computation.*"

Questions to Consider

Is it realistic that queries will only be submitted over the network?

Should the querier be able to set the maximum timeout?

What else should Fuzz do, if anything, to better protect against covert channel attacks?