Imperial College London

COMPUTATIONAL PRIVACY GROUP

Privacy Engineering

Week 3 - Query-based systems

Anonymization is hard for small data and probably impossible for big data

Anonymization needs to:

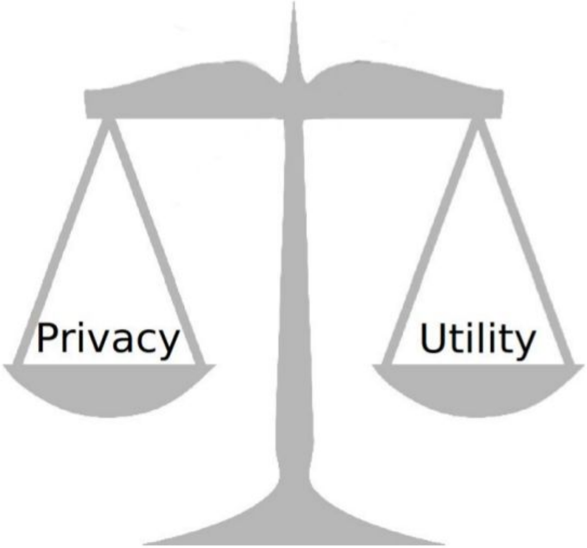
1) protect a dataset against a whole range of attacks: uniqueness, homogeneity, semantic, skewness, matching (unicity), profiling

2) by anonymizing it once and only once

3) all the while preserving utility (all current and future uses).

Hard for small datasets, probably impossible for big data datasets.

Still we need to be able to use datasets anonymously. How do we solve this conundrum?



What if we were to not share the data?

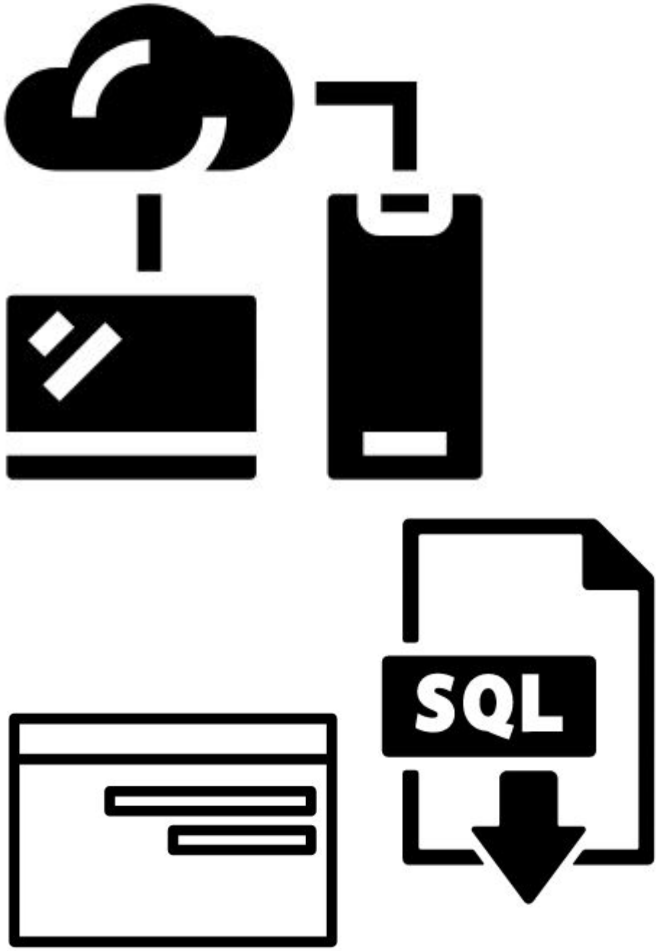
But… that’s the goal?

Actually, not exactly. The goal of anonymous data is to let you **use the data for statistical purposes** (in aggregate).

But why don’t we directly give aggregates to people then?

Query-based systems aim at giving researchers and organizations **anonymous access to the data without sharing with them the individual-level data.**

This can be through online interfaces, SQL queries, verified algorithms, etc. which would only return aggregated data.



Running example: Notepad?

Suppose that we send the following survey to all DoC students:

- What’s your gender?
- What’s your date of birth?
- Which text editor do you use to code?

After pseudonymization the data would look like:

cfcd208495d565	M	1994-09-23	Notepad
1f71e393b38091	F	1993-02-17	Vim
de03beffeed9da	F	1994-11-05	Emacs
7ca57a9f85a19a	M	1995-05-11	Atom
...	...	...	...

Example of queries

A researcher might e.g. be interested in sending the following queries:

1. “How many students born in 1995 code with Notepad?” ← *counting query*
2. “What is the average age of students who do not code with Notepad?
3. “Among students born in March, is Notepad used more by males or females?

cfcd208495d565	M	1994-09-23	Notepad
1f71e393b38091	F	1993-02-17	Vim
de03beffeed9da	F	1994-11-05	Emacs
7ca57a9f85a19a	M	1995-05-11	Atom
...	...	...	...

Is this safe? Any ways you can think of to attack a system like this?

Uniqueness attacks would still work

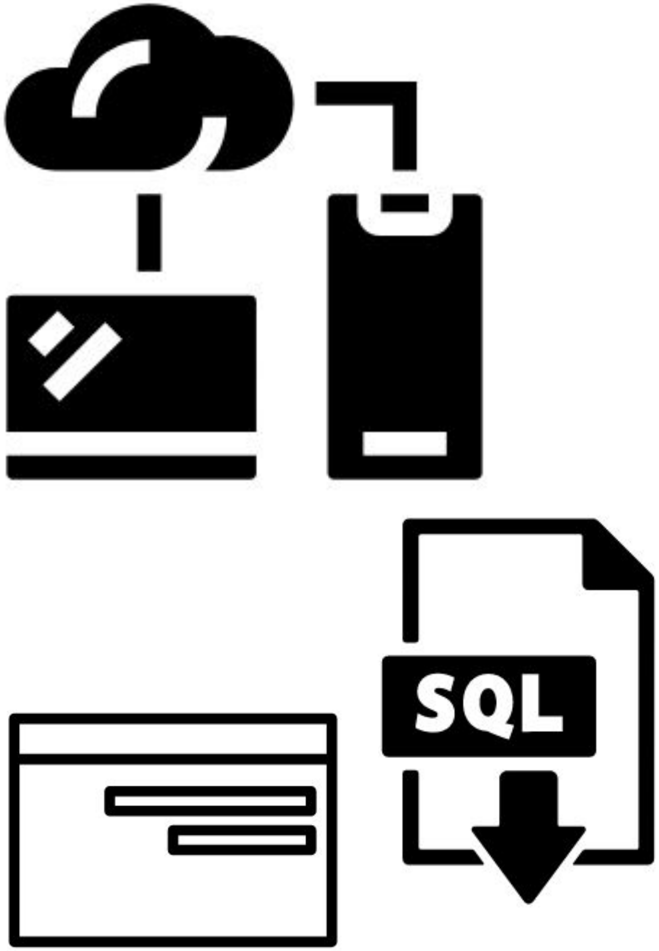
Assume that you know that your course-mate Bob was born on 1994-09-23.

Now you ask the server: “How many students are born on 1994-09-23 and don’t code with Notepad?”

If the answer is 0, then you know for sure that Bob uses Notepad.

Having an interface does not, in itself, prevents basic uniqueness attacks.

How can we prevent this?



Query (set) size restriction (or QSR)

The data holder could try to block such attacks by imposing a **query set size restriction**. The idea is to block any query that selects a "small" set of users.

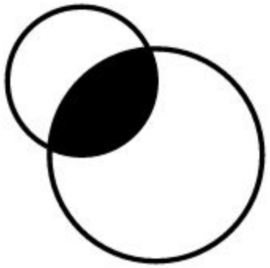
Every query Q includes a logical formula that selects a specific set of users in the dataset D (e.g. `'gender=M AND date_of_birth!=1994-09-23'`). We call this set the query set of Q on D, and we denote it with $\{Q\}_D$ (or just $\{Q\}$).

QSR imposes that every query such that $|\{Q\}_D| <$ threshold is blocked. The choice of the threshold is made by the data curator. For example, he could decide that every query with query set size smaller than 10 returns: `Error: query set too small.`

For a database D, we denote by Q(D) the result of executing Q on D. Observe that if Q is a counting query, then the size of $\{Q\}_D$ is simply Q(D).

Are we safe now? Any ways you can think of to attacks this system again?

Intersection attacks



But then consider the following two counting queries:

- Q: “How many students in DoC code with Notepad”?
- Q’: “How many students in DoC who are not born on 1994-09-23 code with Notepad?”

Both Q and Q’ are likely to have answers *A,A’* >10, so they won’t be blocked by the query set size restriction.

However, if *A*-*A’*=0, Bob doesn’t code with Notepad.

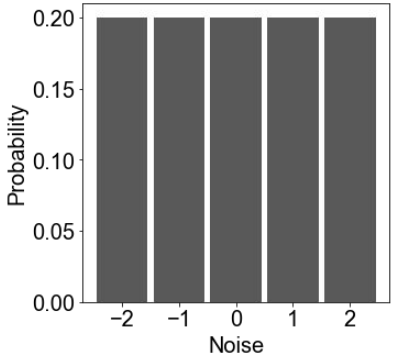
Intersection (also called trackers) attacks **use the answers to multiple queries to learn information about a single individual**

It has been shown that detecting any intersection attacks is an NP-hard problem (see J. M. Kleinberg et al., Auditing Boolean Attributes, PODS 2000)

What else could we do to protect people’s data?

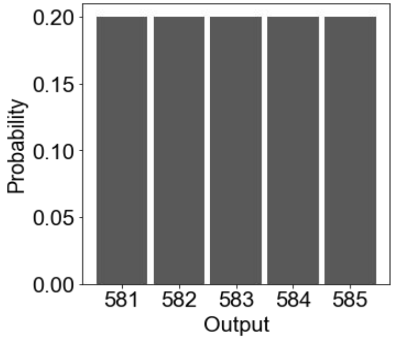
Noise addition (Bounded)

Instead of blocking any queries, we can try to protect privacy by **perturbing the output of every query**. This is usually done by adding some small random noise to the true value of the query before sending the answer back to the user.



For example, we can decide that we add noise randomly selected between -2 and +2.

If the true value of the query is *A*=583, the noisy version will be *Ã* = *A* + noise.



Observe that the noise is centered at 0, so that the expected value of *Ã* is *A*. This means that *Ã* is not *biased*.

Limits of bounded noise addition

Consider our two queries again:

- Q1: “How many students in DoC code with Notepad”?
- Q2: “How many students who are not born on 1994-09-23 code with Notepad?”

Suppose that $\tilde{A}$ = 584+2 = 586 and $\tilde{A}'$ = 583-2 = 581.

The attacker knows that the difference between *A* and *A’* is at most 1 (because she knows that Bob is the only person in the dataset born on that day) and that the noise is bounded between [-2,2] .

She therefore now knows **with certainty** that *A*=584 and *A’*=583 and that Bob uses notepad.

Can we assume that the attacker doesn’t know the noise mechanism we use?
Any other attack against bounded noise you can come up with?

Limits of bounded noise addition: Groups

An attacker can have auxiliary information on multiple people.

For example, she could know that:

- There are 10 male DoC students born in April 1994
- They all use the same text editor

How would you now attack this system?

She can now send the query “How many male students born in April 1994 code with Notepad?”

If none of them use notepad, $A=0$ and $\tilde{A}=[-2,2]$.

If all of them use notepad, $A=10$ and $\tilde{A}=[8,12]$.

The attacker can then know with certainty whether they all use notepad or not.

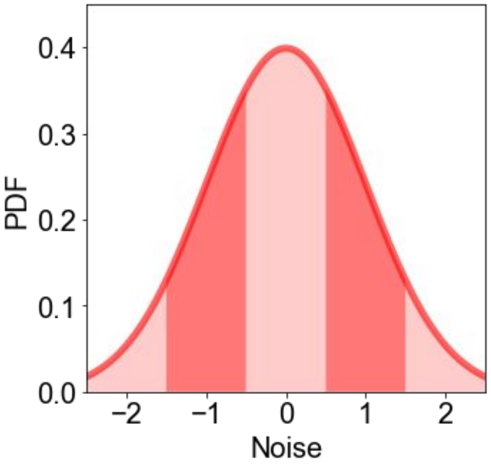
Unbounded noise

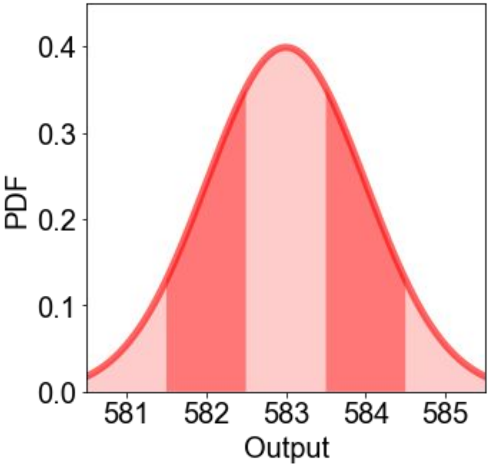
As you probably guessed, the solution to these two attacks is to add **unbounded** noise to the answer.

To limit the impact of noise on utility, we'll however ensure that:

- It is centered at 0, as to not introduce biases.
- Large perturbations are possible (preventing the two previous attacks) but unlikely.

For instance, $\tilde{A} = A + N(0,1)$ which will preserve utility while resisting previous attacks.





Is unbounded noise THE solution or can you imagine ways to attack the system again?

Unbounded noise and Bayes’ theorem

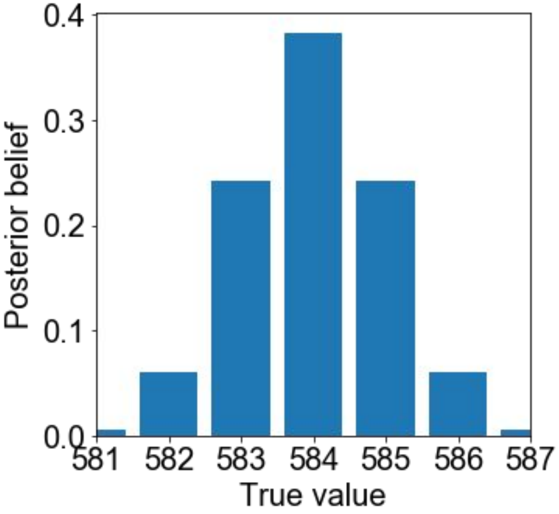
The attacker knows that we’re adding Gaussian noise centered in 0 with a variance of 1 and that *Ã*=584 (true value, *A*=583).

While she can’t know for sure what the true value is, she can still make a guess.

More specifically, she can use Bayes’ rule to compute the *posterior belief* (assuming the true value is between 0 and 1000, the *prior*):

$$\Pr[true = x \mid output = 584] = \frac{\Pr[output = 584 \mid true = x] \times \Pr[true = x]}{\Pr[output = 584]}$$

The result is a (discretized) normal distribution, **centered in 584** and truncated outside [0,1000].



As we start with no information (non informative prior), the only evidence we have (584) is (as expected) our best guess

But now what if we can ask multiple questions?

While the system has to allow people to ask multiple queries (as it'd kill utility otherwise), we add independent Gaussian noise with σ = 1 to all queries.

However, nothing then prevent the attacker from **asking the same question several times** and take the average to find the right value. These are called **averaging attacks**.

Intuitively, the more the attacker asks the same question, the more samples she collects and the more accurate is her estimate.

Formally, this can be done in two ways (frequentist and Bayesian):

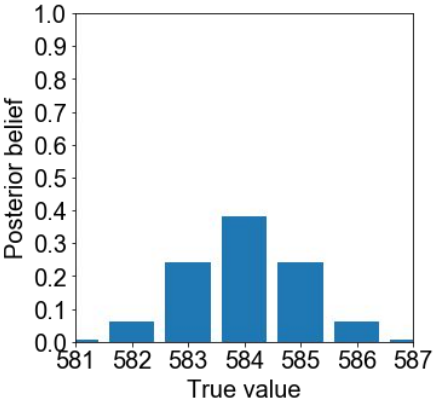
A. Compute the average of the samples and apply the central limit theorem (CLT). CLT says that this average converges to the mean (i.e. the true value).
B. Use Bayes' rule with multiple observations. This method immediately gives not only the most likely value (the average), but also a full posterior distribution.

Averaging attacks: Example

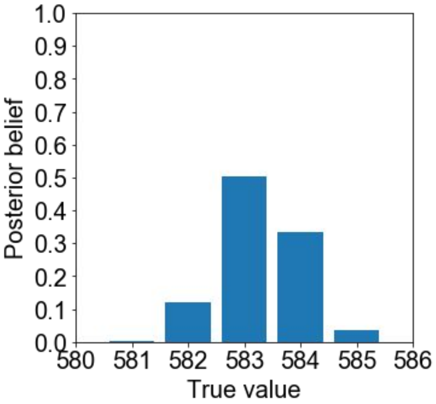
We now run the (Bayesian) averaging attack to discover:

- "How many students in DoC code with Notepad?"

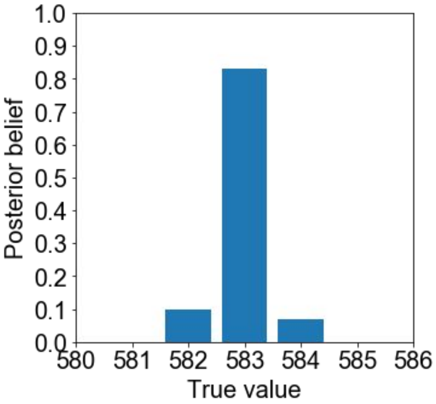
which can then be combined with other queries in an intersection attack to learn whether Bob uses Notepad. Recall that the true value is 583, but the attacker knows only that it's a value between 0 and 1000.



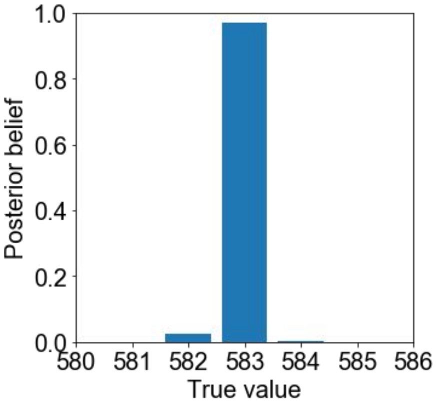
1 query



2 queries



5 queries

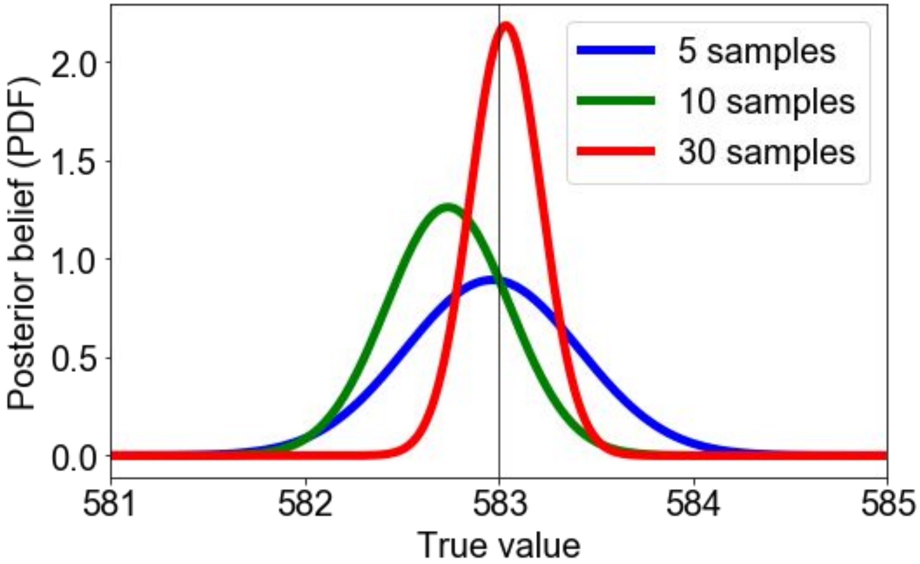


10 queries

Averaging attacks: Example

Plotting the (continuous) distributions allows us to better observe how, as we collect more evidence (queries):

- the mean of the distribution gets closer to the true value (583) and
- the variance (our uncertainty on the true value) decreases



So averaging attacks basically allow us to cancel out the noise that was added by the system, noise-free answers can then be combined in intersection attacks to learn sensitive information. What else could we do to protect the data?

Averaging attacks: how many queries?

Assume your QBS adds independent Gaussian noise $N_i \sim Normal(0, \sigma^2)$.

An attacker performs *n* times the query `COUNT(user=Bob)`. How many queries does he need to check if Bob is in the dataset?

Answer: if $\mathbf{n \geq 4 \sigma^2 z_\alpha^2}$ the attacker will have confidence (probability of making an incorrect prediction) α.

(where $\mathbf{z_\alpha}$ is the (1-α)-percentile of the standard Gaussian)

Typical values: $\mathbf{z_{0.05}=1.96}$, $\mathbf{z_{0.01}=2.58}$

A query $X_i \sim N(0, \sigma^2)$ if Bob is not in the data, and $\sim N(1,\sigma^2)$ otherwise.

He decides that Bob is in the data if $M = \sum_i X_i / n > ½$.

Error probability:
type I: $E_I = P[M > ½ \mid \text{Bob is not in}] = P[M > ½]$ with $M \sim N(0, \sigma^2/N)$
type II: $E_{II} = P[M < ½ \mid \text{Bob is in}] = P[M < ½]$ with $M \sim N(1, \sigma^2/N)$

$\Rightarrow E_I = E_{II} = P[M > ½], M \sim N(0, \sigma^2/n)$

$\Rightarrow P[Error] = E_I\, P[Bob\ in] + E_{II}\, P[Bob\ not] = E_I = P[Z > ½ sqrt(n)/\sigma]$ (with $Z \sim N(0,1)$)

For error probability < α, we then choose the percentile (1-α) of the standard Gaussian, z_α:
$½\ sqrt(n)/\sigma > z_\alpha$

Adding consistent noise: caching and seeded PRNG

The previous attacks only works because we can ask multiple times the same query and get different noise every time. If our noises were to be **consistent,** we wouldn’t learn anything by asking the same question again.

Consistent noise can be achieved through:

- Caching, basically making sure that we cache the noisy result ($\tilde{A}$) of every query and return this if the same question is asked again.
- Seeded pseudorandom number generator (PRNG): in short, we seed our noise generator to ensure that when the query is the same, the exact same noise is added. The seed could e.g. be the hash of all the parameters of the query.

Same question: any remaining attacks you can think of?

Semantic averaging attacks

Adding consistent noise prevents basic averaging attacks, attacks that send the exact same query multiple times.

However, if the query language we use is **expressive enough**, there often exist multiple ways to ask the same questions, e.g. on our dataset we could ask:

- “How many male students born in April 1994 code with Notepad?”
- “How many non-female students born between March and May 1994 code with Notepad?”

Both of them would return the same answer but with different noise (bypassing the cache or seeded PRNG), making an averaging attack possible again.

Semantic attacks can employ queries that are logically equivalent (e.g. the two above), but also queries whose meanings are "similar enough".

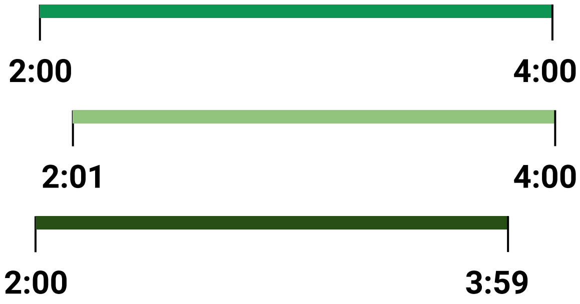
Can you explain why we have to assume that the language has (to some extent) to be expressive? Why are semantic attacks hard to prevent?

Semantic averaging attacks: time-shifting

Time-shifting attacks are an example of an approximate semantic attack. An attack that doesn't return the same value *A* but values that are likely to be close enough

For example, queries could ask for the number of people in a given classroom C:

- From 2:00 to 4:00pm
- From 2:01 to 4:00pm
- But also from 2:00 to 3:59pm, etc



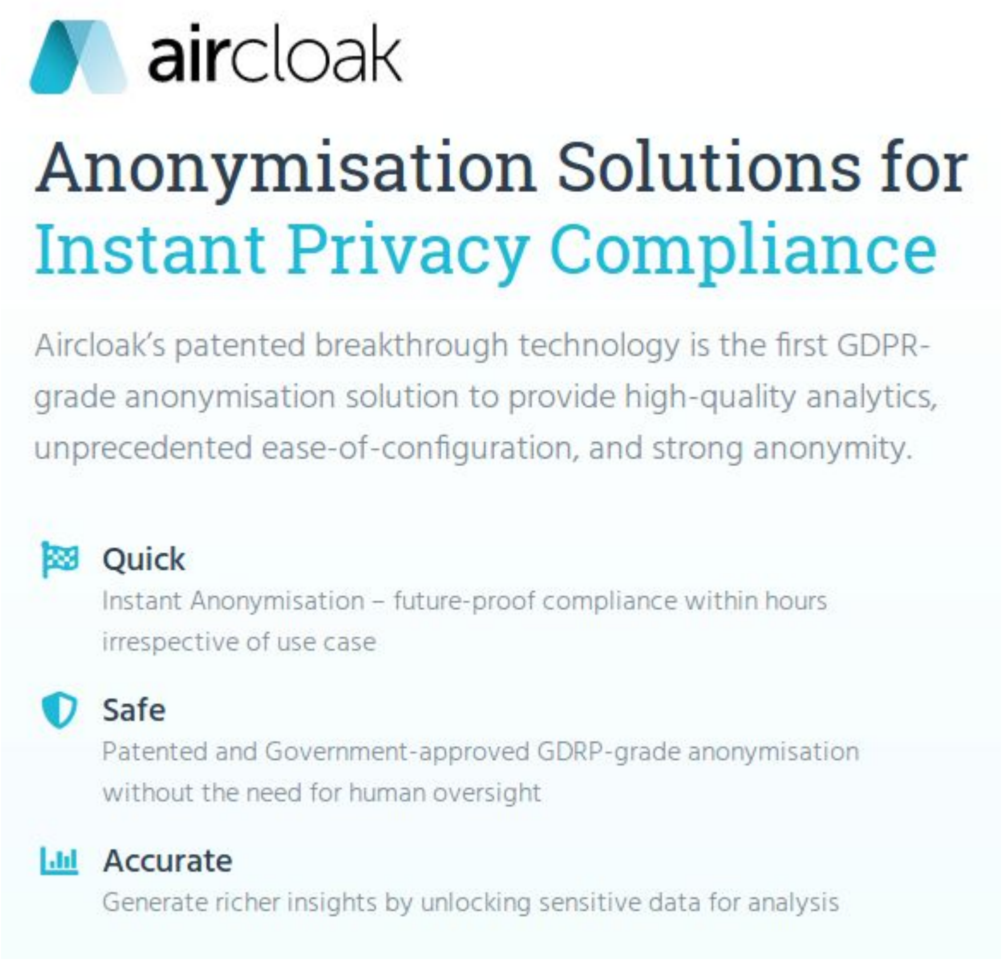
While these might not give the exact same answer, it might still be close enough for an attacker to make a really good guess on the exact number of people who attended the Privacy Engineering class.

No perfect solution here, why?

Hard but not impossible? Meet Diffix

Developed by German start-up Aircloak ($1.3M+ in funding) and the Max-Planck Institute, Diffix is a “*GDPR-grade*” anonymization query-based system providing accurate answers.

Diffix gives analysts an SQL interface adding “sticky noise” (seeded PRNG) to each request.



aircloak

Anonymisation Solutions for Instant Privacy Compliance

Aircloak’s patented breakthrough technology is the first GDPR-grade anonymisation solution to provide high-quality analytics, unprecedented ease-of-configuration, and strong anonymity.

Quick
Instant Anonymisation – future-proof compliance within hours irrespective of use case

Safe
Patented and Government-approved GDRP-grade anonymisation without the need for human oversight

Accurate
Generate richer insights by unlocking sensitive data for analysis

Diffix's sticky noise

For simplicity, we focus only on counting queries. Diffix supports queries of the form:

$$Q \equiv \text{count}(C_1 \wedge \ldots \wedge C_h)$$

where:

- every condition C_i is an expression of the form "attribute □ value";
- □ can be =, ,≠, ≤, <, ≥, or >.

For each condition, Diffix adds two sticky noise values: static noise and dynamic noise. Diffix's output on Q is then:

$$\widetilde{Q}(D) = Q(D) + \sum_{i=1}^{h} \text{static}[C_i] + \sum_{i=1}^{h} \text{dynamic}_Q[C_i]$$

Static and dynamic noise (a bit simplified)

Static noise. For each condition *C*, the noise value static[*C*] is generated drawing a random value from N(0,1). The random value is generated using a PRNG seeded with:

$$\texttt{static\_seed}_{C} = \texttt{hash}(C,\ \texttt{salt})$$

Observe that this seed depends only on *C*, **not** on the query Q.

Dynamic noise. The noise value $\text{dynamic}_Q[C]$ is generated in the same way, but using a different seed that depends also on the query set {Q}. Let {Q} = $\{\text{uid}_1,\ldots,\text{uid}_n\}$. Then:

$$\texttt{dynamic\_seed}_{C,Q} = \texttt{hash}(\texttt{static\_seed}_{C},\ \texttt{uid}_1,\ldots,\texttt{uid}_n)$$

Observe that this seed depends both on *C* **and** on the entire query Q (when selecting $\{\text{uid}_1,\ldots,\text{uid}_n\}$)!

Diffix's bucket suppression (QSR)

Besides sticky noise, Diffix includes another protection. This is called bucket suppression, and it is a more sophisticated version of QSR.

The idea is the same: block any query that selects a set of users with size smaller than a certain threshold. However, the threshold is not fixed, but noisy as well.

If Q(D) ≤ 1, then the query gets suppressed. If Q(D) > 1, then Diffix draws a noisy threshold T from N(4, 1/2), using the seed:

$$\texttt{threshold\_seed} = \texttt{hash}(\texttt{uid}_1, \ldots, \texttt{uid}_n, \texttt{salt})$$

If Q(D) < T, the query is suppressed; otherwise, the noisy output $\widetilde{Q}(D)$ is computed and sent to the analyst.

Note that the threshold depends only on the query set {Q}.

Split averaging attacks

Static noise addition prevent basic averaging attacks, e.g.:
Q: How many students are born in January and code with Notepad?
Q': How many students are *not* born in January and code with Notepad?

Standard noise added to both queries thwarft this attack.

An attacker could, however, submit the same pair of queries with different months:

$$(Q_{Jan},Q'_{Jan}) , (Q_{Feb},Q'_{Feb}) , \ldots, (Q_{Dec},Q'_{Dec})$$

As these queries have different noise, the average of $Q_i(D) + Q_i'(D)$ would converge to the right value.

Diffix prevents split averaging attacks

Diffix’s static noise protects against the split averaging attack. All queries indeed contain the same static noise for the condition “code with Notepad”:

$$\widetilde{Q}_i(D) = Q_i(D) + \text{static}[month = i] + \text{static}[app = Notepad] + \text{dynamic}_{Q'_i}[month = i] + \text{dynamic}_{Q'_i}[app = Notepad]$$

$$\widetilde{Q}'_i(D) = Q'_i(D) + \text{static}[month \neq i] + \text{static}[app = Notepad] + \text{dynamic}_{Q'_i}[month = i] + \text{dynamic}_{Q'_i}[app = Notepad]$$

Which mean that the average of $\{\widetilde{Q}_i(D) + \widetilde{Q}'_i(D)\}_{i=\text{Jan},\ldots,\text{Dec}}$ converges to

tot_notepad + 2*static[`app=Notepad`].

Dynamic noise helps protect against more sophisticated attacks, see https://arxiv.org/abs/1806.02075

Bounty challenge

The developers of Diffix do not have a mathematical proof that their system protects against any attack. And they acknowledge that.

To test Diffix's protections, in Oct 2017 they launched the "Aircloak Privacy Challenge". This is a bug bounty program: researchers and hackers who managed to infer private informations from Diffix's protected datasets (by some definitions) could win money.

The idea behind the challenge is to help Aircloak find new vulnerabilities which they can then fix. This is an example of penetrate-and-patch approach.

This kind of bug bounty programs is very common in computer security, but Aircloak's is the first one for privacy-preserving QBSs.

Three attacks were developed in parallel against Diffix.

Cohen and Nissim's attack

In October 2018, Aloni Cohen and Kobbi Nissim published a report on their attack on Diffix. This attack is an adaptation of a ***reconstruction attack***, proposed in 2003 by Dinur and Nissim.

The reconstruction attack from 2003 aims at reconstructing an entire database, using only counting queries. It is an important attack because it works on any QBS with any perturbation mechanism (e.g. noise addition), provided that:

1. The records have unique IDs and the attacker can use them in the queries.
2. The perturbation to each query is bounded, and this bound is "not too large".

More precisely: Suppose D is a dataset with n records and only one binary attribute. Suppose that the noise added to each query is bounded by B. If B = $o(\sqrt{n})$, then with high probability the dataset can be reconstructed almost entirely in polynomial time.

Pyrgelis, Troncoso, De Cristofaro's attack

In October 2018, some researchers from UCL and EPFL published a blogpost about their attack on Diffix (https://bit.ly/2Q045Dm), based on a previous paper.

This attacks works on location data by acquiring aggregate location time-series indicating the number of people transiting in a certain area at a given time.

Specifically, they take a mobility dataset and extract two subsets D_1 and D_2 that partially overlap. D_1 is used as background knowledge, while D_2 is the protected dataset queried through Diffix.

The goal is to perform a ***membership attack***: given a user in D_1, infer whether the same user is also in D_2.

To do this, the researchers train a classifier on D_1 and then run it on D_2, issuing only queries for aggregate location.

There is no technical report available for this attack yet.

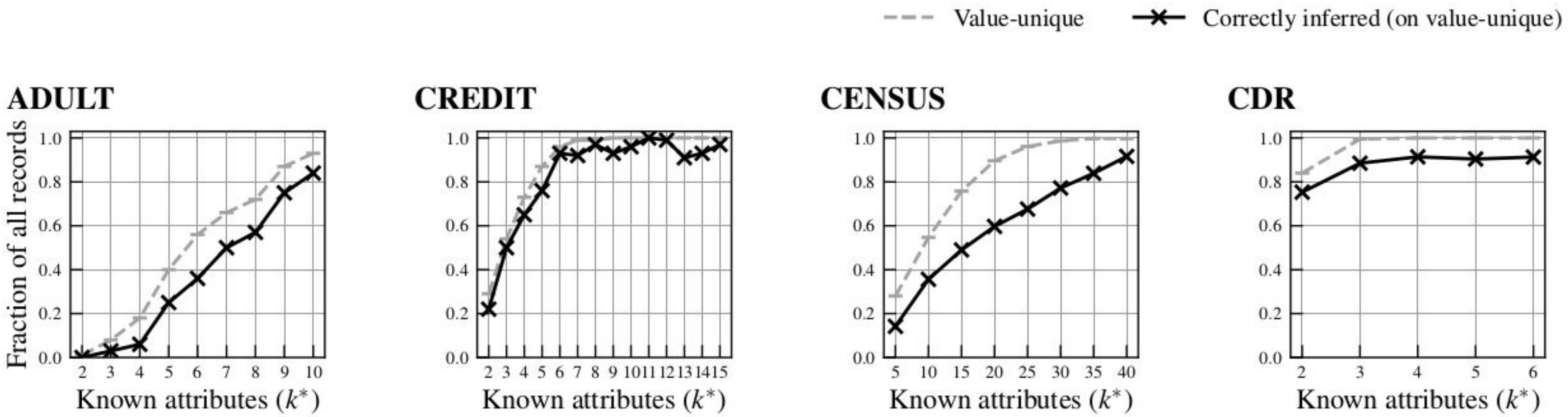
Our noise-exploitation attack

In April 2018, we designed an attack to infer a user's attribute with high accuracy (https://cpg.doc.ic.ac.uk/blog/aircloak-diffix-signal-is-in-the-noise/). We called it a ***noise-exploitation attack***. Here is the intuition:

1. Define queries that are similar enough that they will share part of their static noise. This allows us to cancel out some of Diffix's noise.
2. As Diffix's noise depends on the query set, the noise itself leaks information about the query set. Exploit this to build a likelihood ratio test (LRT).
3. Exploit logical equivalence between queries to circumvent some of the "stickiness" of the noise by repeating (almost) the same query. This produces more samples for the LRT.

It had been previously theorized that data-dependent noise in QBSs could leak information. Our attack is the first example of a noise-exploitation attack on a deployed system.

Diffix only makes re-identification slightly harder



We measure:

- value-unique: fraction of value-unique users
- correctly inferred: fraction of users for which the attack succeeded

- ADULT: US census, 30k records
- CREDIT: CC applications, 690 records
- CENSUS : US census, 200k records
- CDR: synthetic location data for 2M people.

Provable guarantees

In this lecture we have seen several attacks. But there exist many many more attacks on privacy-preserving QBSs.

Attacks can be creative, hard to foresee, and hard to protect against.

To avoid this problem, researchers have been working on privacy-preserving mechanisms that provide mathematical guarantees, and provably protect against (almost) *any* attack.

The main theoretical framework used is called **differential privacy**.

Questions?