Aircloak Anonymization

June 2017

Aircloak's first-in-class real-time anonymized analytics solution provides instant GDPR anonymity compliance and enables high-quality analytics for any data set and any use case. At the core of Aircloak's solution is a unique, patented anonymization technology developed in research partnership between the Max Planck Institute for Software Systems and Aircloak GmbH. This paper describes that technology — how it works, how it impacts the analytics process, and how it protects users in the database.

Summary of Protections

Most of this paper provides a technical description of how Aircloak anonymization works. This first section summarizes those protections in non-technical terms.

Aircloak users "query-by-query" anonymization. Aircloak sits between the analyst and the database, and dynamically examines and modifies both queries and answers.

Aircloak's anonymization is GDPR-compliant

The term "anonymous" is often mis-used or mis-understood to mean weak forms of anonymization like pseudonymization or de-identification. Aircloak is anonymous by GDPR criteria. In fact, CNIL, the French national data protection authority, evaluated Aircloak for a demonstration of compliance. CNIL could not find any vulnerabilities in Aircloak's anonymization protections.

Aircloak uses the EU definition of anonymization

The EU Article 29 opinion[i] on anonymization defines three criteria for anonymization:

- Possibility to single out an individual,
- Possibility to link records relating to an individual, and
- Inference of information concerning an individual.

A system is considered anonymous when the probability of achieving any of these three criteria with high confidence is very small.

Aircloak is anonymous even when an analyst has substantial database knowledge or external knowledge

Often anonymization schemes fail because an attacking analyst is able to exploit externally derived knowledge, or has partial knowledge about the contents of the database. Aircloak's protections hold even when the analyst has full knowledge of up to all but one database column or the equivalent in external knowledge. Aircloak's protections hold when the analyst has partial knowledge of database rows (50% or even more, depending on the size of the database) or the equivalent in external knowledge.

Aircloak prevents analysts from deducing presence or absence of a single user in a query answer

Aircloak adds a small amount of noise to query answers: just enough to hide individual users in the database with high confidence.

Aircloak prevents the release of data that applies to only one or a small number of users

Any value or text output by Aircloak is guaranteed to apply to at least 2 users, and on average applies to at least 5 users.

Aircloak prevents repeated queries from removing noise from answers

Any set of queries that result in identical answers will receive identical noise. This prevents repeated answers from removing the noise through an averaging attack.

Aircloak prevents combinations of queries with different ranges from revealing information about individual users

Attacks using pairs of queries that differ in range enough to include or exclude a single user are defended against.

Aircloak prevents combinations of queries with and without pseudo-identifiers from revealing information about individual users

Attacks using pairs of queries that include or exclude an individual user using a pseudo-identifier for that user are prevented.

Aircloak prevents all known combinations of queries from revealing information about individual users

To Aircloak's and CNIL's knowledge, there are no queries or combinations of queries that can reveal information about individual users.

What is Anonymization?

The word "anonymization", used in its most general sense, simply means making it harder to identify individuals in a data set. There are, however, stronger and weaker forms of anonymization. In this paper, we use the term "anonymization" in a strong sense, for instance as defined by the EU Article 29 opinion and the GDPR. Strong anonymization means that even people with substantial knowledge about individuals in the database cannot identify which data in the database represents those individuals.

By contrast, the EU opinion uses the term "pseudonymization" (also referred to as "de-identification") to refer to the weaker form of anonymization. Pseudonymization typically means only removing Personally Identifying Information (PII) like names and addresses from the database, but otherwise leaving the data intact. An analyst with even just a little knowledge about an individual in a

pseudonymized database can often identify that individual's data (referred to as "re-identification").

Consider the following example of re-identification of pseudonymized data. A telecommunications company with a mobile call-record database containing user phone number, cell-tower location, and time of call, wishes to release the data to an analyst, Bob. The company pseudonymizes the data by replacing the phone numbers with random numbers. Bob notices, however, that the times of several phone calls match those of calls between him and his girlfriend Sally, and the cell-tower locations of those calls correspond to his and her homes. Furthermore, no other calls match this pattern. As a result, Bob knows which call records belong to Sally, and learns about locations that she has visited.

Anonymization is an old, and until Aircloak, largely unsolved problem. The difficulty isn't in making data anonymous per se. Indeed many techniques have been developed over the last 35 years that do just this, for instance K-anonymity, Differential Privacy, and L-diversity just to name a few. Rather the trick is getting both anonymization and high utility. This is where Aircloak's unique technology far out-performs legacy techniques.

Query-by-Query Anonymization

Aircloak's anonymization utilizes a number of basic principles. The first such principle is "query-by-query anonymization". Legacy techniques like K-anonymity anonymize the entire database all at once, and then make the anonymized database fully available to analysts (Figure 1). By contrast, Aircloak anonymizes on a per-query basis (Figure 2).

The difficulty faced by full-database anonymization is that the database must be anonymized for all possible queries that an analyst might make. This leads to aggressive distortion of the data, typically rendering the anonymized data useless. This is why, for example, HIPAA defines the "Safe Harbor" de-identification rules, in spite of the fact they are known not to be anonymous[ii].

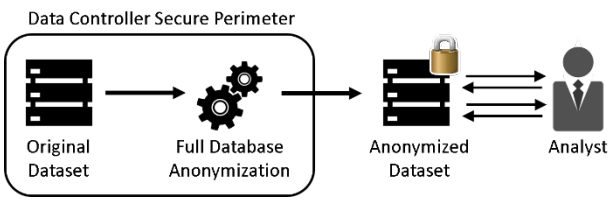


Figure 1: Legacy full-database anonymization requires aggressive data distortion, destroying the data

Query-by-query anonymization allows Aircloak to apply only as much data distortion as is absolutely necessary for the given query. The following sections, each of which describes other basic principles of Aircloak's anonymization, explain how Aircloak minimizes data distortion.

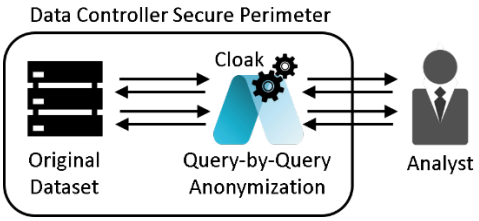


Figure 2: Query-by-query anonymization: The Cloak tailors data distortion to the query, minimizing distortion and increasing utility

Layered Fixed Noise

Aircloak is by no means the first to propose query-by-query anonymization. Differential Privacy, a well-researched theoretical anonymization model, is also query-by-query. The basic idea behind Differential Privacy is that answers are perturbed by zero-mean random noise. However, the random noise can be removed by repeating the same answer many times, and taking the average of the answers. To put a hard bound on theoretical privacy loss, Differential Privacy limits the number of queries an analyst can make (the so-called "budget"). It is primarily this limitation that makes Differential Privacy impractical.

A key benefit of adding noise to answers is that it works independently of the data type. In order to get this benefit without the budget limitation of Differential Privacy, Aircloak invented and patented Layered Fixed Noise. The idea here is that the noise value assigned to a given query is fixed to that query. If the same query is repeated, even with a different syntax, then the same noise is assigned. This way, even if an analyst causes and answer to be repeated, he or she cannot remove the noise.

Aircloak's implementation of Layered Fixed Noise does not require that the Cloak remember all prior answers. Rather, the Cloak examines the query, identifies the query's filter conditions, and uses each filter condition as the basis for a separate layer of noise. In addition, the Cloak examines the set of users that comprise the answer, and generates an additional layer of noise. These layers of noise are summed together to generate the final noise that is applied to the answer.

The key concept here is that if the analyst tries to repeat a query, even with a different syntax, the Cloak will add the same noise and so the noise can't be averaged away.

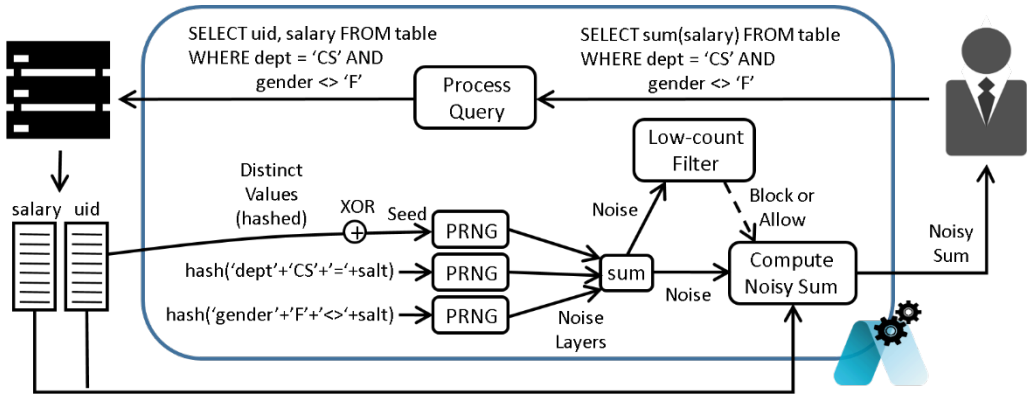


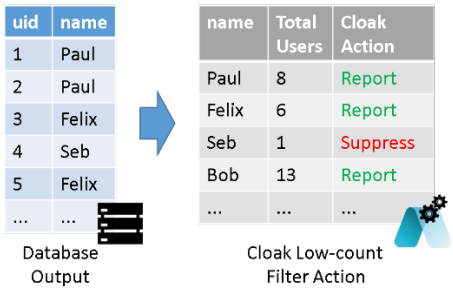
Figure 3: The cloak identifies filter conditions in the query, and adds layers of fixed noise, one per filter condition as well as one for the User ID (UID) column

This is illustrated in Figure 3. The analyst's query has two filter conditions, one on department (dept) and one on gender. The Cloak generates a separate noise layer for each of these conditions based on the "semantics", or intended effect, of the query (i.e. the column name, the comparator, and the column value). The Cloak also generates a noise layer based on the set of distinct UIDs in the query. Each of the noise layers is generated by seeding a Pseudo-Random Noise Generator (PRNG) with the appropriate input (filter conditions or UIDs).

Layered Fixed Noise defends against a variety of sophisticated attacks, described later.

Noisy Low-count Filtering

Aircloak can handle any type of data, including free-form text in any character encoding. This means for instance that it is possible for the analyst to query for a list of first names, or a list of searches that users have made.



uid	name
1	Paul
2	Paul
3	Felix
4	Seb
5	Felix
...	...

Database Output

name	Total Users	Cloak Action
Paul	8	Report
Felix	6	Report
Seb	1	Suppress
Bob	13	Report
...	...	...

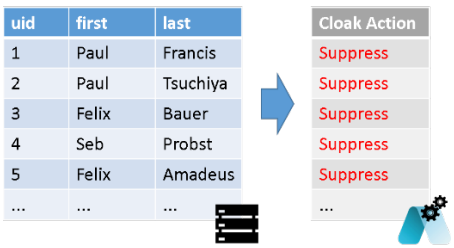
Cloak Low-count Filter Action

Figure 4: Low-count filter: Because very few users have the string "Seb", that string will be suppressed in the output

One of the anonymization properties of Aircloak is that no single data value (text string or number) is released by the cloak unless at least some number of users share that data value. This property is similar to the property intended by K-anonymity: only reveal data if enough users share that data. By way of example, consider Figure 4, which shows the table returned by the database to the Cloak, as well as the Cloak's actions based on that table.

In Figure 4, only one user has the name "Seb". Because this falls below the Low-count Filter threshold, that string will be suppressed, and instead replaced with the star symbol (*). The analyst may know that something was suppressed, but won't know what.

In addition to adding Fixed Noise, this Low-count suppression is another form of data perturbation. Because Aircloak operates query-by-query, however, the amount of distortion is tailored to the given answer. For instance, for the query of Figure 4, the Cloak is able to reveal many first names. Suppose, however, that the analyst asks for both first and last name, as shown in Figure 5. Here, because most combined first and last names are unique, the Cloak will suppress almost everything. By contrast, legacy K-anonymity approaches would have to assume that a future query might request both first and last name, and suppress both from the start (and many other columns as well).



uid	first	last
1	Paul	Francis
2	Paul	Tsuchiya
3	Felix	Bauer
4	Seb	Probst
5	Felix	Amadeus
...	...	...

Cloak Action
Suppress
Suppress
Suppress
Suppress
Suppress
...

Figure 5: Low-count filter: Since the combination of first and last name is unique for almost every user, the Cloak suppresses almost all output

Our implementation of the Low-count Filter is not based on a constant "hard" threshold, but rather on what we call a "soft" Fixed Noisy Threshold. Specifically, the threshold used to decide whether or not to suppress an output is itself a random number from a Gaussian distribution, for instance one with mean 5 and standard deviation 1. In addition, the

Fixed Noisy Threshold noise value is based on Layered Fixed Noise.

The problem with using a hard threshold would be that the simple existence of a reported output tells the analyst something concrete about the data, and might therefore be used as the basis for some clever attack on the system. The Fixed Noisy Threshold adds an additional amount of uncertainty.

Limiting Query Semantics

The correct operation of Fixed Noise depends on the Cloak being able to identify the filter conditions in a query. To ensure that all filter conditions are identifiable, Aircloak places certain limits on the SQL syntax.

Currently disallowed SQL constructs are:

- Most constructs that provide set union semantics (i.e. "OR")
- Arbitrarily fine-grained ranges or numerical constants in conditional clauses (WHERE and HAVING) and certain functions
- The ability for the analyst to write procedural code
- Any math on columns that are used in ranges in conditional clauses
- Any math operators (+, -, /, *, %) in combination with constants and numerical functions that are discontinuous, such as ceil, floor, and div.

Note that many of the SQL constructs that are currently disallowed in the Cloak will be allowed in future releases. The exception is the allowance of procedural language, which opens up attacks that we believe it is prohibitively expensive to defend against.

Constructs with Set Union Semantics

Currently Aircloak disallows most constructs with set union semantics. This includes:

- The OR logical operator (i.e. in WHERE and HAVING clauses)
- The UNION statement
- The CASE statement (WHEN statements that produce the same label have union semantics)

The exception to this is the IN operator, for which we add noise layers (see below).

Fine-grained ranges and numerical constants

Aircloak prevents analysts from specifying arbitrarily fine-grained ranges in filter comparisons. We call this "Fixed-alignment". Fixed-alignment forces numbers in ranges to adhere to a pre-determined set of fixed range sizes and boundaries. For numbers, we currently use range sizes in a sequence of "1-2-5", i.e. 1, 2, 5, 10, 20, 50, 100, 200, 500 etc. This applies for decimal values as well: 0.01, 0.02, 0.05, 0.1, 0.2, 0.5, etc., and negative values. Note that these values correspond to the way many currencies work: 1 cent, 2 cents, 5 cents, 10 cents, 50 cents, 1 euro, 2 euros, 5 euros, etc.

Besides fixing range sizes, Aircloak also fixes the boundaries on which the ranges may lie. Otherwise an analyst could isolate a victim by simply shifting an otherwise fixed-size range in tiny increments. We fix boundaries to the same "1-2-5" values as ranges, as well as 50% shifts of these. By way of example, $10 \leq X < 20$ is allowed (size of 10 fixed to a boundary of 10), and $5 \leq X < 15$ is allowed (size of 10 fixed to a boundary shifted by 50% of 10, which is 5), but $9 \leq X < 19$ is not allowed.

Fixed-alignment also applies to datetime (dates and times) ranges, though with alignments that are natural for datetimes rather than the "1-2-5" sequence — of course seconds, hours, minutes etc., but also natural sub-divisions with each time unit, for instance 1, 2, 5, 10, 15, and 30 seconds.

The LIMIT and OFFSET clauses must also be fixed-aligned, because otherwise the LIMIT and OFFSET could be used to isolate arbitrary users in the system, even without analyst knowledge of those users. LIMIT must be at least 10.

Math Restrictions

In certain cases, math operations (+, -, *, /, %) can be used to mimic other operations that Aircloak either disallows or actively monitors. One of these is the range function, where math is used on the conditional column to scale the target value to match the fixed-aligned value.

Math operations (+, -, *, /, %), combined with non-continuous numerical operations such as floor, ceil, or div can be used to mimic logic (AND, OR, NOT) and comparison operators (=, >, <, >=, <=). If allowed, these would allow the analyst to get around the above restrictions. Without providing a full example, we note for instance that

```
ceil((age - 29) / 100)
```

is equivalent to

age > 29

for numbers less than 100, and (a * b) is equivalent to (a AND b) if a and b are limited to 1's and 0's. Therefore, Aircloak restricts combining math operations with non-continuous functions when used in combination with constant values.

Minimal Amount of Distortion

The basic goal of Aircloak's anonymity is to prevent an analyst from having high confidence about the presence or absence of an individual user in a given answer.

Aircloak currently provides the following anonymized aggregation functions: count, sum, avg, stddev, max, min, and median. To anonymize these functions, there are two basic mechanisms we apply (either individually or together):

1. Add noise to answers.
2. Ensure that any given user is grouped together with other users.

At the same time, we wish to maximize utility. This means minimizing noise as well as the size of groups to the extent possible while maintaining strong anonymity.

Adding Noise

Aircloak adds Gaussian noise to answers. This is also true for the noisy threshold used with the low-count filter.

Aircloak targets a standard deviation (SD) for fixed noise of roughly two times the expected contribution of a single user to the answer. The noise is slightly less for queries with very few conditional clauses, and somewhat more for queries with many conditional clauses (more than four).

To better understand what this noise level means, consider a query that counts the total number of users in the database. In this case, each user contributes one to the total count. Therefore, we target a SD of roughly 2. We refer to queries that count the number of users as a "counting query".

On the other hand, consider a query that counts the total number of hospital stays. Suppose that the patients with the most stays have roughly 20 stays. To hide these patients, the Cloak would target a SD of two times this, or roughly 40.

What does this mean in terms of an analyst's confidence about the presence or absence of a given user? With SD=2, the probability that a counting query gives the exact answer is around 20% (noting that noise is rounded to the nearest integer). The probability that the answer is off by one is around 35%, and off by two, 24%. As a result, in the absence of additional knowledge, the analyst can say virtually nothing about an individual user.

Suppose, however, that the analyst somehow knows that the answer to a counting query can only be one of two possible exact adjacent values (i.e. 24 if the victim does not have an attribute, and 25 if he or she does). If the noisy answer is 24 or 25, there is only a 53% chance that the noisy answer is the correct answer. If the answer is 23, there is a 59% chance that the answer is 24, and if 22, there is a 65% chance that the answer is 24. The confidence in the correct answer grows as the noise grows.

On the other hand, the probability of getting high noise in a Gaussian distribution is small. For instance, if the noisy answer is 15, there is a 91% chance that the correct answer is 24, but the probability of so much noise happens only roughly once in 100000 answers.

Grouping Users

Besides adding noise proportional to the largest answer contributions from users, Aircloak also masks the effect of outliers (where an outlier is defined here simply as one of the several highest data points, rather than as an abnormal data point per se).

To mask outliers, Aircloak takes the highest few users (using a noisy threshold), and modifies their values with the average of the subsequent highest users.

Aggregation Functions

To compute count and sum, Aircloak masks the outliers, and then computes the resulting count or sum. Aircloak then takes the average of the next largest group of users and uses this average to set the SD of the noise. This noise is added to the computed count or sum.

For avg, Aircloak simply uses the anonymized count and sum functions (avg = sum / count). To compute stddev, Aircloak computes the true variance of each row, and then computes the anonymized avg of the true variances.

Unlike count, sum, avg, and stddev, the functions min, max, and median do not add noise per se. Rather, after outlier masking (for min and max), these functions take the average of a small group of users around the true median, min, or max, and report that value. Indeed, these functions may provide an exact

answer as long as the group of users all share that value.

Examples of Attacks

Up to now we've described *what* Aircloak does, but not so much *why*. The rest of the paper explains the why by describing a number of attacks and explaining how Aircloak defends against them.

More sophisticated averaging attacks

Aircloak defends against a naïve averaging attack where the analyst repeats a given query.

Syntax-based averaging attack: An analyst could try a somewhat more sophisticated averaging attack where the *semantics* of each query stays the same, but the *syntax* changes. An example would be the following sequence of filter conditions:

```
salary = 100000
salary + 1 = 100001
salary + 2 = 100002
.....
```

While the syntax changes with each new condition, the effect of each condition is the same. Because Aircloak seeds its noise layers on the semantics of the filter condition, these filter conditions would all produce the same noise. Besides this, the UID-based noise layer would also remain the same with each query.

Incremental range-change averaging attack: Another form of averaging attack would be to produce a sequence of ranges that differ semantically, but never-the-less produce the same result. For instance

```
salary BETWEEN 0 AND 100000
salary BETWEEN 0 AND 100000.0001
salary BETWEEN 0 AND 100000.0002
.....
```

The semantics of each condition changes, but the answer does not. The fixed-alignment of ranges prevents the analyst from generating these filter conditions. Here as well the noise layer from the UIDs would also prevent averaging.

Modified UID-list based averaging attack

The analyst could try forcing the UID list to differ with each answer so that the noise layer from the UIDs could be averaged away. For instance, consider the following sequence of filter condition *pairs*:

Pair 1:

```
salary = 100000 AND age = 20
salary = 100000 AND age <> 20
```

Pair 2:

```
salary = 100000 AND age = 21
salary = 100000 AND age <> 21
```

....

Each pair, taken together, lists all users with salary = 100000. Each individual answer, however, contains a potentially different set of UIDs. As a result the UID-seeded noise layer changes each time and can be averaged away. The layer due to the 'salary = 100000' filter condition, however, remains the same and is not averaged away, thus defeating the attack.

Difference Attacks

There is a class of attacks we call the "difference attack" that would enabled by Fixed Noise if there were only one layer of noise based on the UIDs. The idea behind this attack would be to create a pair of queries whose answers are either identical, or differ by only a single user, and try to use the corresponding answers to determine which is the case.

By way of example, consider the two queries of Figure 6. Assume in this case that the analyst knows there is only a single person in the database with the job title of CEO. The answer to Query 1 contains all patients diagnosed with AIDS, but excludes the CEO. The answer to Query 2 contains all patients with AIDS. If the CEO has AIDS, then he or she will appear in the answer to Query 2, and therefore the fixed noise for the UID-based noise layer would be different if the CEO has AIDS.

Query 1

```
SELECT count(*)
FROM diseases JOIN patients
WHERE disease = 'AIDS' AND
      job_title <> 'CEO'
```

Query 2

```
SELECT count(*)
FROM diseases JOIN patients
WHERE disease = 'AIDS'
```

Figure 6: Difference Attack: The answers for these two queries will differ if and only if the CEO has AIDS. Layered Fixed Noise prevents this attack.

The queries of Figure 6 are said to "isolate" the CEO, who we refer to as the "victim" of the attack.

This example explains the need for filter-condition noise layers in addition to the UID noise layer. Query 1 requires a filter condition that doesn't exist in Query 2. As a result, the answers are very likely to differ based on the differing noise layers alone: the mere fact that the noisy answers differ does not say anything about whether or not the underlying true answers differ.

There are a variety of ways that an analyst might try a difference attack. For instance, Figure 7 shows a difference attack where the analyst knows that the

victim is the only user with a salary in the range of $150000 to $200000. Both of these queries are properly fixed aligned, and so would not be prevented. In this case, Query 2 definitely excludes the victim, while Query 1 includes the victim is he or she has AIDS.

Query 1

```
SELECT count(*)
FROM diseases JOIN patients
WHERE t.disease = 'AIDS' AND
        t.mod_salary >= 100000 AND
        t.mod_salary < 200000
```

Query 2

```
SELECT count(*)
FROM diseases JOIN patients
WHERE t.disease = 'AIDS' AND
        t.mod_salary >= 100000 AND
        t.mod_salary < 150000
```

Figure 7: Difference attack exploiting fixed-aligned ranges in the special case where the victim is the only user with salary between $150000 and $200000. Layered Fixed Noise prevents this attack.

This attack is prevented because the noise layer derived from the range would differ between the two queries, forcing the answers to (likely) be different whether or not the underlying true answers are the same or not.

SQL's LIKE function allows queries to do partial matching on text strings. For instance, [name LIKE 'Mar_' would match the names 'Mary', 'Mari', 'Marg', and so on. [name LIKE 'A%'] would match all names beginning with 'A' ('Alex', 'Andrew', 'Allison', etc.). LIKE is a very powerful string matching function, and Aircloak allows it.

A difference attack using LIKE is possible in cases where the group of users that match two LIKE expressions differ by one user. For example, suppose that the lastname column of a table has multiple 'Smith' names, but only one 'Smithson' and no other names that start with 'Smith'. In this case, the queries shown in Figure 8 can be used to isolate the user named 'Smithson'.

Query 1

```
SELECT count(*)
FROM diseases JOIN patients
WHERE disease = 'AIDS' AND
        lastname LIKE 'Smith%'
```

Query 2

```
SELECT count(*)
FROM diseases JOIN patients
WHERE disease = 'AIDS' AND
        lastname = 'Smith'
```

Figure 8: Difference attack where database has multiple 'Smith's, one 'Smithson', and no other names starting with 'Smith'. Layered Fixed Noise prevents this attack.

To defend against this kind of attack, Aircloak adds noise layers for each wildcard character ('%' or '_').

Queries with complex syntax

An analyst may wish to prevent the filter condition layers through complex syntax that the Cloak does not understand. An example would be the following query:

```
SELECT count(*), age_30_or_40 FROM (
 SELECT
 uid,
 (age_30 + age_40) % 2 as age_30_or_40 FROM (
  SELECT
  uid,
  floor((age_greater_29 + age_less_31) / 2) AS age_30,
  floor((age_greater_39 + age_less_41) / 2) AS age_40
  FROM (
   SELECT
   uid,
   ceil((age - 29) / 100) AS age_greater_29,
   ceil(0 - (age - 31) / 100) AS age_less_31,
   ceil((age - 39) / 100) AS age_greater_39,
   ceil(0 - (age - 41) / 100) AS age_less_41
   FROM players
  ) x
 ) y
) z
GROUP BY age_30_or_40;
```

Although this query has for instance no WHERE clause, it is semantically identical to the following:

```
SELECT count(*)
FROM players
WHERE age = 30 OR age = 40
```

The Cloak prevents the first query due to the math restrictions it places on query syntax.

Remaining Attacks

The set of attacks described above is not comprehensive, but is representative of the major classes of attacks an analyst could make. With the exception of the attacks and constraints described in this section, Aircloak is not aware of any attacks that can break anonymity— that is, allow an attacker to obtain a high-confidence guess regarding any given user's attribute values with high probability.

A pre-requisite for Aircloak's anonymity is that each real user is represented by a single UID in the database. This condition can be violated if, for instance, the system that inserts new users into the database doesn't check for this condition. It is the responsibility of the data provider to ensure that UIDs are unique per user.

Perfectly correlated user groups: Aircloak noise levels are set so as to protect individual users in the database. If a group of users behaves identically in such a way that can be exploited by the analyst, then Aircloak only defends against if the the noise levels

and low-count threshold levels are increased proportionally. An example would be a case where, say, 10 users (including the victim) all exclusively visit the same places in a geolocation database. With its default noise levels, the Cloak would not hide the movement of these 10 users, and the analyst could make conclusions about the victim if the analyst knows in advance that the 10 users are so correlated.

Dynamically changing databases: Though planned for an upcoming release, currently Aircloak does not defend against difference attacks that exploit predictable changes in the database. For instance, suppose that the analyst makes the following query on consecutive days:

```
SELECT count(*)
FROM employees
WHERE dept = 'CS' AND
        salary = 100000
```

Suppose further that the analyst knows that a single given person (the victim) was added to the database after the first query but before the second. The Cloak would add identical noise layers if the victim is in neither answer, but the UID noise layer would differ if the victim has a salary of 100000 and is therefore in the latter answer.

Aircloak can be currently used in environments where the analyst is very unlikely to have the external knowledge needed to exploit this attack.

Summary

Aircloak's patented anonymization technology is unprecedented in the strength of anonymity and high level of utility it offers. Aircloak's query-by-query approach, combining Layered Fixed Noise, Low-count Filtering, Fixed-alignment, and SQL restrictions, is a breakthrough that will transform the way organizations share data.

Aircloak is committed to working with Data Protection Authorities and Data Privacy Officers to obtain approvals for anonymized data sharing. Please contact us for more information on how Aircloak can help you with your private analytics needs.

www.aircloak.com
solutions@aircloak.com

[i] EU Article 29 Data Protection Working Party "Opinion 05/2014 on Anonymisation Techniques", (URL)

[ii] "Guidance Regarding Methods for De-identification of Protected Health Information in Accordance with the HIPAA Privacy Rule," (URL).