Aircloak Anonymization

Aircloak's first-in-class real-time anonymized analytics solution provides instant privacy compliance and enables high-quality analytics for any data set and any use case. At the core of Aircloak's solution is our unique, patent-pending anonymization technology. This proprietary 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. It also serves as a roadmap to the rest of the document.

Aircloak users "query-by-query" anonymization. Aircloak sits between the analyst and the database, and dynamically examines and modifies both queries and answers. (See section *Query-by-Query Anonymization* on page 2.)

Aircloak's anonymization is strong

The term "anonymous" is often mis-used or mis-understood to mean weak forms of anonymization like pseudonymization or de-identification. Aircloak is anonymous in the strongest sense of the word. (See section *What is Anonymization?* on page 1.)

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. (See section *Minimal Amount of Distortion* on page 7.)

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. (See section *Noisy Low-count Filtering* on page 3.)

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 process. (See section *Fixed Noise* on page 2.)

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

Pairs of queries that differ in range enough to include or exclude a single user are actively detected and prevented. (See sections *Fine-grained ranges and numerical constants* on page 4 and *Active Defenses* on page 6.)

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

Pairs of queries that include or exclude an individual user using a pseudo-identifier for that user are actively detected and prevented. (See sections *Constructs with Set Union Semantics* on page 4 and *Active Defenses* on page 6.)

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

To Aircloak's knowledge, there are no pairs of combinations of queries that can reveal information about individual users. An exception exists if the SQL LIKE function is enabled. In this case, there are theoretical cases where extremely rare combinations of text in the database would allow an analyst to deduce information about an individual user. In our tests on databases ranging in size from 1000 to 500,000 distinct users, no such cases existed. (See section *Remaining Attacks* on page 8.)

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[i]. 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 [i] 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 calls that she has made to other people.

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. By contrast, Aircloak anonymizes on a per-query basis.

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].

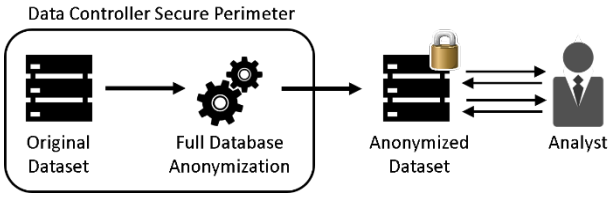

Figure 1: Legacy full-database anonymization requires aggressive data distortion, destroying the data

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 exactly how Aircloak minimizes data distortion.

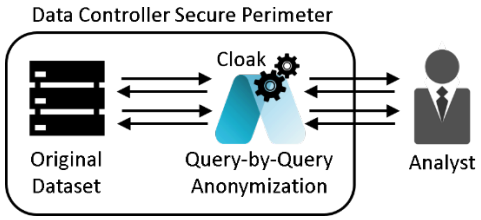

Figure 2: Query-by-query anonymization: The Cloak tailors data distortion to the query, minimizing distortion and increasing utility

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. 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 limitation of Differential Privacy, Aircloak invented Fixed Noise. The idea here is that the noise value assigned to an answer is fixed to that answer. If the same answer is produced, then the same noise is assigned. This way, even if an analyst causes and answer to be repeated, he or she can never remove the noise.

Aircloak's implementation of Fixed Noise does not require that the Cloak remember all prior answers. Rather, the Cloak modifies queries so that, in addition

to the information requested by the analyst, the Cloak also obtains from the database the user identifiers (UIDs) associated with the information. These UIDs are then used as the basis for seeding the random number generator used to generate the noise (Figure 3). All answers with the same underlying set of users will generate the same noise.

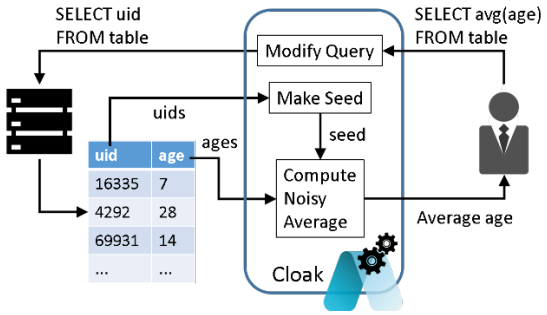

Figure 3: Fixed Noise: Aircloak modifies the query to include the User ID (uid), which in turn seeds the noise random number generator

As Figure 3 shows, anonymization done by the cloak takes place on a table retrieved from the database that contains at least a UID column. The UID column is used by the cloak in various ways to protect individuals (as identified by the UID) in the database. Fixed noise is only one of several anonymization mechanisms that use the UID.

Noisy Low-count Filtering

Aircloak can handle any type of data, including arbitrary text. 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.

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.

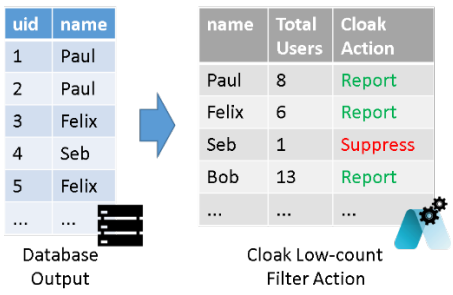

Figure 4: Low-count filter: Because very few users have the string "Seb", that string will be suppressed in the output

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 will 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).

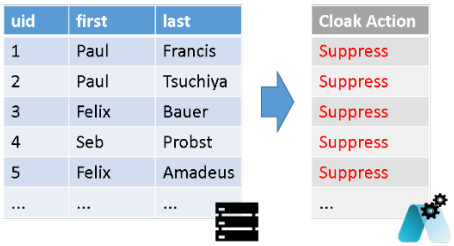

Figure 5: Low-count filter: Since the combination of first and last name is unique for almost every user, the Cloak suppresses almost all output

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 is fixed to the set of distinct users in the answer, as used with 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 Queries

Although Fixed Noise prevents the simple "repeated answer" attack that Differential Privacy is susceptible to, it opens the door to another kind of attack that we call the "Difference Attack". This attack stems from the simple observation that, if two answers

given by the Cloak are different, then the data used to generate those two answers must necessarily also be different. Otherwise, the Cloak would have returned the same answer.

A malicious analyst could try to exploit this fact to learn something about an individual user by creating two queries whose answers differ by at most a single user. By way of example, consider the two queries of Figure 6. The answer to Query 1 will contain all patients diagnosed with AIDS, as well as the patient named Paul Francis living at 123 Elm St. The answer to Query 2 will contain all patients with AIDS. If this particular Paul Francis does not have AIDS, then he will not appear in the answer to Query 2, and therefore the fixed noise will be seeded differently. Therefore, if the two answers differ, then this particular Paul Francis definitely does not have AIDS (or, more precisely, the database definitely does not indicate that Paul Francis has AIDS). Note that if the two answers are the same, this is not definitive proof that Paul Francis does have AIDS, because there is a small probability that the noise may have randomly produced the same answer.

Query 1

```
SELECT count(*)
FROM diseases JOIN patients
WHERE disease = 'AIDS' OR
   (name = 'Paul Francis' AND
    addr = '123 Elm St.')
```

Query 2

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

Figure 6: Difference Attack: The answers for these two queries will differ if and only if the user Paul Francis at 123 Elm St. does not have AIDS (note that Query 1 is currently disallowed by the Cloak because of the "OR")

There are broadly two ways that Aircloak prevents difference attacks. The first is to disallow certain SQL constructs that could lead to a difference attack. These restrictions, in combination with Fixed Noise and Low-count Filtering provide very strong anonymity protections.

Never-the-less, some low-probability attacks remain possible, and so an optional second method is to actively check queries and answers to ensure that a low probability difference attack is not taking place. The former is described here, and the latter in the next section (Actively Filtering for Difference Attacks on page 5).

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 (+, -, /, *, %) associated with 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, as we develop mechanisms and software for actively detecting difference attacks. 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, which is actively filtered for difference attacks (see below).

Fine-grained ranges and numerical constants

The queries of Figure 6 are said to "isolate" the user Paul Francis, who we refer to as the "victim" of the attack. This case of isolation is prevented by disallowing the OR operator.

Isolation is also in principle possible using a range in the where clause. For instance, consider the attack of Figure 7. If the victim is the only user in the database with a salary of $136558 (and the analyst knows this), then the analyst can isolate the victim with the queries shown. Here, Query 1 includes the victim if and only if the victim has AIDS, and Query 2 excludes the victim for sure. In this attack, if the two queries differ, then the analyst knows that the victim definitely has AIDS.

Query 1

```
SELECT count(*)
FROM diseases JOIN patients
WHERE disease = 'AIDS' AND
      salary > 136557
```

Query 2

```
SELECT count(*)
FROM diseases JOIN patients
WHERE disease = 'AIDS' AND
      salary > 136558
```

Figure 7: Difference attack using WHERE clause ranges. Here the victim is the only user with a salary of $136557 (note that these queries are currently disallowed by the Cloak because of Fixed-alignment)

Aircloak invented a mechanism called "Fixed-alignment" to prevent this kind of attack. 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. Note that these values correspond to how 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 disallows the queries of Figure 7. Rather, the analyst would have to make a query for instance like one of those shown in Figure 8. It is highly unlikely that one of these queries can be used to isolate the victim of Figure 7. (Note that the rare case where the victim could be isolated is detected and prevented with active filtering, described later on.)

```
SELECT count(*)
FROM diseases JOIN patients
WHERE disease = 'AIDS' AND
      salary >= 130000 AND
      salary < 140000
```

```
SELECT count(*)
FROM diseases JOIN patients
WHERE disease = 'AIDS' AND
      salary >= 100000 AND
      salary < 200000
```

Figure 8: Fixed-alignment: The query of Figure 7 is disallowed. Rather, a fixed-aligned query such as one of these must be used instead.

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 (see Figure 9.)

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 and the active filtering described in the following section. 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.

Query 1

```
SELECT count(*) FROM
  (SELECT disease,
   salary / 1.36558 AS mod_salary
   FROM diseases JOIN patients) t
WHERE t.disease = 'AIDS' AND
        t.mod_salary >= 100000 AND
        t.mod_salary < 200000
```

Query 2

```
SELECT count(*) FROM
  (SELECT disease,
   salary / 1.36559 AS mod_salary
   FROM diseases JOIN patients) t
WHERE t.disease = 'AIDS' AND
        t.mod_salary >= 100000 AND
        t.mod_salary < 200000
```

Figure 9: A math function on the column used for the range can be used to get around fixed-alignment (note this use of math is disallowed by the Cloak)

Actively Filtering for Difference Attacks

The mechanisms and restrictions described up to this point provide very strong protections. Never-the-less, there may exist some rare cases where an analyst can still succeed in launching a difference attack. As a result, Aircloak provides mechanisms that defend against almost all of these rare cases.

Rare Conditions

For a difference attack (without active filtering) to be possible, the following conditions must all hold:

1. The user being attacked (the victim) must have a single isolating attribute.
2. The analyst must know the attribute.
3. The analyst must be highly confident that the attribute is isolating.

An isolating attribute is a value (i.e. text, number, or datetime) that is unique to the user. Examples of isolating attributes are [job_title = 'CEO'], [date_of_birth = '1922-12-03'] or [salary = 136559]. Note that it must be a single attribute that isolates the user: a combination of attributes like (date_of_birth AND zip_code AND gender), which typically would be enough to isolate a user, cannot be used because of the restrictions Aircloak places on SQL.

To give a concrete example of what these rare conditions mean, let's consider date of birth because it is often found in databases, and because one's date of birth can easily be learned. Suppose a database contains 100,000 persons. There are roughly 36,500 possible birthdates among people 100 years old or less. This means that, on average, each birthdate has almost three individuals (2.74). Of course, even disregarding very old people, many people in the database will have a unique date of birth. But in the absence of specific knowledge about all of the birthdates in the database, the analyst has no confidence that any given (non-old) individual has a unique birthdate, and therefore cannot use it to isolate and attack victims.

Even isolating relatively old people using birthdate is hard. According to the US census, approximately 1.4% of the population is between 85 and 94. From our database of 100,000 users, this is 1400 users spread over 3650 birthdates. As a result, there is only a roughly 62% probability that any such old person has a unique birthdate. This is not a high enough confidence to isolate a victim in a difference attack. Indeed users would have to be over 95 years old for the analyst to have a 95% confidence of being isolated. On the other hand, the chances that the analyst happens to know a 95 year old person in the database is quite small.

Active Defenses

Figure 10 shows a difference attack on the victim isolated with the job title of 'CEO'. In Query 1, the victim is definitely excluded from the answer. In Query 2, the victim is excluded only if he or she does not have AIDS. If the answers to the two queries differ, then the victim definitely 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 10: Difference attack using "not equal". This attack is actively detected and prevented.

To defend against this attack (and others described below), the Cloak examines the effect of each component of the WHERE clause on the overall answer. In particular, it identifies the rows that would differ if either of the components were removed from the query. If the differing rows are themselves low-count, then the corresponding WHERE clause component is removed from the query.

In this case, the Cloak would recognize that [job_title <> 'CEO'] makes only a low-count difference in the answer, and so would remove it. With respect to the attack, this would make Query 1 identical to Query 2, and the analyst would learn nothing.

Figure 11 shows another difference attack on the victim isolated with the job title of 'CEO'. This attack uses the IN clause, which is currently the only allowed operation that has union semantics. In Query 2, the victim is definitely not in the answer. In Query 1, the victim is in the answer if he or she has AIDS. If the answers differ, then the victim definitely has AIDS.

Query 1

```
SELECT count(*)
FROM diseases JOIN patients
WHERE disease = 'AIDS' AND
   job_title IN ('dev', 'sales', 'CEO')
```

Query 2

```
SELECT count(*)
FROM diseases JOIN patients
WHERE disease = 'AIDS' AND
   job_title IN ('dev', 'sales')
```

Figure 11: Difference attack using victim in IN clause. This attack is actively detected and prevented.

To defend against this attack, the Cloak examines the impact of each constant in the IN clause, and removes any that have a low-count impact.

In the difference attack of Figure 12, the IN clause is used to enunciate every job title except CEO (effectively making the semantics equivalent to Query 1 of Figure 10). Here the Cloak checks the impact of the entire IN clause for low-count.

Query 1

```
SELECT count(*)
FROM diseases JOIN patients
WHERE disease = 'AIDS' AND
   job_title IN ('dev', 'sales',
      'marketing', 'design', ...., )
```

Query 2

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

Figure 12: Difference attack using IN to enunciate all job titles except CEO. This attack is actively detected and prevented.

Even with fixed-alignment, there may be cases where a user may still be isolated with ranges. In the

absence of active detection, the attack of Figure 13 would work if the victim is the only user with a salary in the range of $150000 to $200000 (and the attacker knows this). 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 13: Difference attack exploiting fixed-aligned ranges in the special case where the victim is the only user with salary between $150000 and $200000. This attack is actively detected and prevented.

To defend against this, Aircloak has invented a mechanism called "shrink-and-drop". The basic idea is that, any time a query includes a range, the Cloak checks possible valid smaller fixed ranges to see if those ranges would result in a low-count number of users being isolated. If they would be, then the corresponding rows are removed from the answer.

Figure 14 illustrates a difference attack that can only be done under additional special conditions, namely that all users in the answer share an attribute except one. In this case, the analyst knows that there is only one woman in the development department.

Query 1

```
SELECT count(*)
FROM diseases JOIN patients
WHERE disease = 'AIDS' AND
      dept = 'dev' AND
      gender = 'F'
```

Query 2

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

Figure 14: Difference attack exploiting the case where there is only one woman in the development department. This attack is actively detected and prevented

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 sets the standard deviation (SD) for fixed noise to be roughly two times the expected contribution of a single user to the answer. To better understand what we mean by this, 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 set SD = 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 average patient has 4 hospital stays. Since in this case one patient influences the answer by 4 (on average), we set SD=8.

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 right 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 average answer contribution 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). This is

necessary because we use average contribution of users to generate noise rather than the worst case contribution of users.

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 remaining 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.

Remaining Attacks

The mechanisms and restrictions described above protect against the vast majority of possible attacks. However, it does not protect against every conceivable attack.

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 this is the case.

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 analyst increases the noise levels and low-count threshold levels. 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.

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 15 can be used to isolate the user named 'Smithson'.

Figure 15: Difference attack where database has multiple 'Smith's, one 'Smithson', and no other names starting with 'Smith'

Of course, the LIKE function may be disabled if the data controller believes that it is necessary. Alternatively, the analyst could disable access to any column that might be attackable. However, the likelihood of the conditions for this attack appearing in practice is very small.

To see why, suppose that the analyst knows that there is a user with last name Smithson in the database. Smith is (according to one source) 335 times more common than Smithson. The analyst can query the database to see how many Smith's there are. If the number of Smith's is roughly 150 or less, the analyst can be somewhat confident that there is only one Smithson in the database. But there are also a number of other last names that start with 'Smith' (Smitham, Smithee, Smithers, and so on). These names would collectively have to be very improbable for the analyst to be confident that none of them also appear in the database.

To test this, Aircloak examined three different types of text fields: surnames, search terms, and Twitter hash tags. The surnames were synthetically generated according to the frequency of surnames published by the US census. The search and Twitter hash tags are from real databases. In tables ranging from 1000 to 500,000 distinct users, not a single instance of an attackable text field existed.

Summary

Aircloak's patent-pending anonymization technology is unprecedented in the strength of anonymity and high level of utility it offers. Aircloak's query-by-query approach, combining Fixed Noise, Low-count Filtering, Fixed-alignment, SQL restrictions, and active monitoring of queries, 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).