INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES CENTRE VAL DE LOIRE

CHEMAK Ramy
4A
STI

Academic Year :
2019 / 2020

« Analysis and contribution to the GDA (General Data Anonymity) Score project »

GDA Score | General Data Anonymity

Laboratory Internship supervisor:

NGUYEN Benjamin
Senior Researcher

Teacher supervisor:

EICHLER Cédric

LABORATOIRE D'INFORMATIQUE LIFO FONDAMENTALE D'ORLEANS

Computer Science Laboratory of Orléans

Bourges 18000

Acknoledgement

This report presents my work during this research internship. This would wouldn't have been achieved without the help of several people though.

I would like to thank **Pr. Cédric Eichler** and **Pr. Benjamin Nguyen** for offering me this work opportunity. The internship wouldn't take place without them.

I would also like to thank **M. Paul Francis** from the Max Planck Institute and his assistant **M. Mohammadali Forouzesh** for their technical support all along the internship.

Abstract

Despite the exponential growth of data collection during the recent years, data privacy remains a major concern for users. Information about us is being constantly collected through every connected device we use. Some of this data is very personal and highly sensitive. It must be protected. With the introduction of new legislations, data de-identification came under highlight more than ever.

This research internship's main task is to develop and contribute to the General Data Anonymity (GDA) Score software. This report gives an overview the gda-score framework and the main related-work realised. It also presents Diffix's anonymisation mechanism and its affiliation with gda-score. And finally, the report compares between GDA Score and ARX de-identifier, another anonymisation tool.

These tasks were all carried out, with different levels of success. Some issues were encountered during the internship, both logistic and scientific. Some satisfying results were achieved however. Some of the tasks were left unfinished, but the report describes precisely their advancement so they can be resumed later on.

Keywords: Anonymisation, Data privacy, Database, De-identification software

Contents

Introduction

The development of Big Data technologies during recent has led to the increase of large-scale personal data collection. This collection is conducted by both, public and private actors. Thus, the protection of this data is becoming even more important, with the exponential increase of massive data collection. Many scandals however over leaks of large amount of personal data brought this issue into spotlight. [Examples ...] These scandals pushed the European Union to take futher measures to tackle by voting the General Data Protection Regulation (GDPR).

The GDPR requires from all personal data holders to ensure the anonymisation of the datasets at their disposal in order to fullfill a number of privacy criteria. It makes it clear that pseudonymisation isn't enough match theses criteria. Anonymised data embedded on a query-based system must be of a minimum resilience against typical re-identification attacks. The main technique for data protection and privacy preservation known so far relies on data anonymisation. Nevertheless, German researchers affiliated with Max Planck Institute for Software Systems and Aircloak recently introduced ***Diffix***. Diffix is is promoted to be a commercial solution to address the data privacy issue raised by the GDPR. Diffix's resilience is challenged though by a new class of attacks claimed to be efficient againt the system. The same team of reaserchers also developed the General Data Anonymity (GDA) score tool, in order to provide assessments for their anonymisation system, as well as other known systems.

The internship's mission is part of the ongoing effort to address the data privacy issue. More specifically, carry an analysis of the GDA score tool and contribute to its development and improvment. Other tasks were also carried out, regarding the analysis of Diffix's sticky noise and the development of the ARX software.

The first chapter regards data privacy litterature. It presents the state of the art in the field. It also introduces more specific knowledge regarding Diffix and its attack. The second chapter introduces the GDA Score framework, and describes its main features. The third chapter reports the done work in relation with gda-score API. It describes the implemented attacks, results and analysis. The fourth and last chapter present work done regarding ARX development. It presents some clues and tracks for further implementation of software features.

Chapter I - Data privacy: State of the Art

Before boarding on the main mission which concerns data privacy, I was then lacking some perquisites. Therefore, the first few days were mainly dedicated to compensate this lack, get an overview of the existing literature and fade in within the most relevant notions.

1.1. Importance of anonymity

With the democratisation of data collection techniques, big data technologies and data mining, it became almost too easy to build up large databases. The latter are very valuable for business marketing, data analysts and scientists, for their statistics. In the medical field for example, researchers need these databases to conduct analysis in the aim of foreseeing some disease-related tendencies or other. Those databases are however build up form personal data. Giving how critical someone's personal medical record can be, it's obvious that these data need to be protected. Therefore, we turn to data anonymisation techniques to ensure individuals personal privacy.

1.2. Regulations

in the aftermath of the second world war, a common awareness of importance of personal privacy has developed. Many people become conscious of the sensitivity of private personal details for many individuals.

In France, the very first law addressing the issue is the "Informatics and Liberty" act, voted in 1978. Among other things, it explicitly introduces the notions of personal information, identification of an information and anonymity of information. It requires the protection of private information by all necessary means from possible de-anonymisation.

The leak of huge amount of data concerning millions of individuals during the past years pushed the political power in different countries to adopt legislative initiations in a try to limit the of these consecutive scandals. Many of these leaks concerned European citizens while the companies responsible for it where extra-European.

This led the EU to vote the GDPR in 2018. The regulation introduces precise definitions for identifiable information, sensitive information, means to be employed for privacy protection and many other technical notions. It makes clear distinction between anonymisation and pseudonymisation. The regulation introduces as well some very heavy financial sanctions against data holders who wouldn't abide by the law and save effort to protect individuals privacy.

The various related laws voted across the world, whether the GDPR or other national acts, ignited the data privacy experts to push researches in the field, improve and optimise already existing technologies, and come with new ones which would better answer the increasing requistries from the public opinion and the political power.

1.3. Data anonymisation

1.3.1. De-anonymisation and attacks

Let's consider an anonymised database with a number of anonymised records, each initially corresponds to a particular individual. Some of the attributes in the de-identified dataset are actually public. For example, if in real life we are able to know each individual's gender, then we consider the attribute "gender" public, even when it's kept in the anonymised dataset. It can be deen as sort of background knowledge.

We consider that a journalist knows all the public attributes for each individual present in the dataset. A journalist will try to de-anonymise all individuals of the dataset. So the risk of the journalist consists in the risk of re-identifying each individual.

The model of the attorney assumes that an attorney knows all public attributes of one particular individual present in the dataset. It is this one single individual he's targeting. The risk of the attorney consists in the risk of re-identifying this individual. Nevertheless, depending on each individual, the re-identification risk value is different. Among those values, the risk on the attorney corresponds to the maximum risk value.

The model of the marketer makes the same hypothesis as the model of the journalist. The marketer does not terget a specific individual, but aims at de-identifying a large fraction of records within the dataset.

1.3.2. De-identification techniques

Quasi-Identifier and attributes types

Within a record, we usually remove identifying attributes such as names or social security numbers. However, some combinations of other attributes can be used for re-identification attacks, such as address and bith date and place. Such combination is known as a Quasi-Identifier (QID).
The main purpose of data anonymisation is to protect sensitive attributes which disclosure can be harmful for individuals. Diseases are typical examples of sensitive attributes. We might have though some attributes which are insensitive, and which values can be kept unmodified.

K-Anonymity

K-Anonymity is one of the most common privacy models. We consider that a dataset is k-anonymous if each record is undistinguishable from at least k-1 other records regarding the considered QID for the dataset. Such group of k look-alike records is called an equivalence class.

L-Diversity

This privacy model is used to protect sensitive attributes from risks of disclosure. We consider that an equivalence class respects the l-deiversity criteria if, for some sensitive attribute, we have l distinct values. The criteria validation can be generalised for the whole dataset, only if it's valid for each equivalence class.

1.3.3. Differential privacy

Unlike the previous criteria, differential privacy isn't a privacy model. It's rather a measure of data processing algorithms. It requires, for a particular anonymisation process, that the output dataset isn't significantly altered when adding or removing an individual's record from the input dataset.
Differential privacy is supposed to guarantee that information about a specific individualscannot be leaked or disclosed just regarding its membership. This guarantee however might fade if the attacker dispose of some background knowledge.

1.4. Diffix

1.4.1. Description

Diffix operates more as an SQL proxy between the analyst and a live database. The analyst can be a live human or an data extracting program.

Sticky noise

The purpose of Diffix is to not only to add noise to the answer, but also make it dependent on the data. Therefore, Diffix uses the notion of sticky noise. Two types of noise are used, static and dynamic. Noise layers are computed based on conditions passed in the SQL query.
A uid-seed value is generated using a Pseudo Random Number Generator (PRNG) based on the XOR of distinct uids hashes. The point of this uid-seed is to neutralise average attacks because the noise won't alter for multiple queries returning the same answer.

The condition-related dynamic noise is based on the seed of a PRNG with a hash of the condition. Diffix adds a secret salt as well which is query independt. We can issue the hypothesis that this salt is a commercial data, which might explains the lack of further details about it.

Sticky threshold

The threshold is a small number, which if returned distinct uid count is below, the answer is supressed. This value is determined by the mean of a number K. It's similar to query set size restriction. The default implementation sets K=4. But the threshold is computed based on K, while taking into consideration the number of noise layers.

Researches haves already proven that a fixed threshold is not enough for data privacy guarantees. For a hard-coded threshold K_f, we have the situation where an analyst can trigger two queries with two output counts of distinct users K_f and K_f+1. Giving the eventual difference introduced between the two queries, the analyst would be able to deduce some guesses with confidence.

Diffix addresses this issue through bucket suppression. A static filter require that if the query's output count is lower or equal than 2, then the query is automatically suppressed. Otherwise, Diffix computes a noisy threshold T, based on a Gaussian distribution, with K as a parameter, and using the seed defined in the previous section. If the raw query output count is strictly lower than T, then the query is suppressed. Otherwise, the output is processed by Diffix, in order to generate the noisy output to be returned to the analyst.

1.4.2. Tests and results

The GDA Score framework (see Chapter II) comes with a number of databases hosted on a PostgreSQL server. Via the gda-score API, we can query both raw and Diffix-anonymised and compare results to see how much noise is added.

The selected dataset for test is table "loans" from database "banking" (see section 2.3.4). The test consists in querying the table, with and without triggering Diffix's proxy, and noting down the results for analysis. To keep it simple yet explicit, sent queries use statistical functions which would return a single numerical value. It will make comparison and analysis clearer.

Query	DB	Result
SELECT count(*) FROM loans WHERE gender="Male"	Raw DB	410.0
	Anon DB	410.0
SELECT count(*) FROM loans WHERE gender="Female"	Raw DB	417.0
	Anon DB	417.0
SELECT avg(duration) FROM loans WHERE gender="Male"	Raw DB	35,684
	Anon DB	35,615
SELECT avg(duration) FROM loans WHERE gender="Female"	Raw DB	36.863
	Anon DB	36.897

Figure 1: Diffix's sticky noise test (1)

We can observe throughout the first table, that the first two queries didn't show any significant noise or difference. The third and fourth queries put into practice however this added noise. This noise seems quite slight though, and that can be expected. It can be explained by the lack of value's diversity for the attribute "duration" on one hand. On the other hand, the number of records which the query is averaging is low. This hypothesis is based on raw observation the dataset hosted on the PostgreSQL server. Diffix's sticky noise also depends on the condition clause, whereas here we have only one condition. This might also justify the small amount of noise added.

Moreover, we can consider the small noise appeared in the two last queries from one side, and the fact that the returned value for the two first from the side is a truncated float (we have for example a value which looks like 417.0 instead of the integer 417). This observation highly promotes the hypothesis that if no noise is observed, that's because it was too small to notice after rounding the result.

For the second test, we slightly edit the queries by editing the requested results. It's very important to notice that the condition clause hasn't been changed.

Query	DB	Result
SELECT avg(duration), count(*) FROM loans WHERE gender="Male"	Raw DB	35,684; 410.0
	Anon DB	35,615; 410.0
SELECT avg(duration), count(*) FROM loans WHERE gender="Female"	Raw DB	36.863; 417.0
	Anon DB	36.897; 417.0

Figure 2: Diffix's sticky noise test (2)

When observing the numerical results, we notice that they are unchanged. Whether it's the raw answers or the added noise, both are actually the same. This allows us to verify that indeed, the sticky noise depends on the condition clause only, via the algorithm described in previous sections.

For the third test, we use the first test's queries as a base, and we delete the condition clause.

Query	DB	Result
SELECT avg(duration) FROM loans	Raw DB	36.26118
	Anon DB	36.27376
SELECT count(*) FROM loans	Raw DB	827.0
	Anon DB	826.0

Figure 3: Diffix's sticky noise test (3)

Once again, the average query explicitly shows a difference in the numerical result between the raw and the Diffix-anonymised queries. This difference corresponds to the expected sticky noise. What is more interesting, is that this time we can observe the noise for count request as well. The sum of the count for the raw queries during the first test matches the result for this test (410+417=827). When checking the sum for the Diffix-anonymised results, we numbers don't add up due to this noise. This validates the hypothesis made in the analysis of the first test set.

Another element to notice is that, despite the suppression of the condition clause, we can still observe the noise. This puts into evidence the secret salt previously mentioned in the Diffix description.

1.4.3. Noise-exploitation attack

Noise-exploitation attack is a class of attacks introduced by a group of researchers from the Computational Privacy Group of Imperial College London (CPG-ICL). It aims at taking advantage of the Diffix structure in order to infer individual's sensitive attributes in the dataset. The CPG-ICL researchers developed two varieties of attacks: differential and cloning attacks.

The principle behind this kind of attack is that if the noise is dependant on the data itself, then it presents the risk of information leakage. As mentioned previously, Diffix adds sticky noise composed of some static noise based on the query and dynamic noise based on the raw returned answer. The attack seeks to learn information about the raw answer using the dynamic noise, neutralise part of this noise using multiple queries, and logically equivalent queries to cancel the seed which generates the dynamic noise.

For both attacks, we would be working on a set of attributes A, which can be extended with a sensitive attribute s. We temporarily make the assumption that s is a binary attribute, ie. it has only two possible values. As in the paper, we will consider 0 and 1 as possible values for the sensitive attribute, to simplify the examples. We also make the hypothesis that we already have prior knowledge of all values for attributes set A for the record we're looking to infer.

Differential noise-exploitation attack

The differential noise-exploitation tries to infer both values of the sensitive attribute, each carried by a sample of queries. Then it processes the results in order to discriminate between the two distributions relying on probability calculations. The attack takes place in two steps. The first is selection and validation of data. The second is applying the very differential attack algorithm, in a try to infer the sensitive attribute.

The attack assumes that the record which sensitive attribute we are trying to infer is unique. This means that for its values combination of A, there is no other record in the dataset with exactly the same values for A. Therefore, we assume that there's an oracle Unique which would take the defined prior knowledge as an input and ensure that it uniquely identifies the targeted record.

For the first step, we start by selecting a subset of A of size $k < |A|$. All along the attack, we will be actually iterating over a huge number of subsets for all possible values of k. Once the subset is selected, we process it through the oracle Unique. If the oracle validates it, then we proceed to the second step. Otherwise, we proceed to the next subset.

The second step is to process the selected subset through the differential attack algorithm. To carry this task on, we alter different combinations of condition clause for queries. We fix the sensitive attribute condition to s=0. The other attributes from A also are included in the condition set with all of them set to their truthful value, except for one which value is altered in the condition in order to have two queries. We count then the query output set size for both queries, and if they are both strictly positive, we save the difference. The attribute to be used for query duplication is itself altered regularly to generate a test sample big enough for discrimination later. We repeat the same process after changing the sensitive attribute condition to s=1. We also generate a list of difference values for this case. We end up with two lists of difference values, one for each possible value for s.
The case both lists are empty corresponds to the situation where no samples are available to carry out the probability calculations properly. Otherwise, we use Gaussian distribution functions to compute the probability for each value of s to be true. Hereafter, we can guess our sensitive attribute's real value, and the inference attack is done.

We can notice that the attack take advantage of some particular aspects of Diffix's mechanism. The sticky noise is an aggregation of noises generated by each condition present in the condition's clause. The attack keeps this condition clause almost unchanged, and mutates one condition each time. This might allow averaging the overall generated noise, and thus, cancelling a part of it. Besides, for each couple of queries generated to compute the so-called difference, we change a small difference in the query, which is that value of the altered attribute from A. This will change the seed used for the dynamic noise, due to the change of the returned raw answer. Therefore, we can learn some about the information.

Cloning noise-exploitation attack

The cloning noise-exploitation attack can be seen as an extension to the differential attack. Instead of altering conditions related to the attacker's background knowledge, it rather uses dummy conditions which wouldn't affect the query's user output. And it's these conditions which we play with in order to have the samples to work on. Just like the differential attack, it proceeds in two steps, one for the preliminaries and the second for the inference algorithm process.

The attack assumes that queries used for statistics aren't bucket suppressed. It also makes the assumption that, regardless of the sensitive attribute's value, the targeted user is unique-value according to the set A and its already known values. This means that no other record possesses the same values for the set A. The dummy conditions set, The most crucial ingredient for the attack, is to be generated before the overall attack starts.

The first step, we select a subset of A of size k-1, with $k < |A|$. We ensure, first that this subset is unique-value, and second that queries are going through the bucket suppression protection. All along the attack, we will be actually iterating over a huge number of subsets for all possible values of k. Once both conditions are validated, we process this subset and the dummy conditions set through the cloning attack algorithm. Otherwise, we skip to the next subset. If no subset fulfils the requirements, then the dataset is considered non attackable.

Unlike the differential attack, the cloning attack requires from the attacker which value is to be inferred. We assume for our example that we are trying to infer s=0. The second step is similar to the differential attack

when it comes to generating a list of so-called difference values. We fix the s-related condition to s<>0, with a certain combination of other dummy conditions. We measure the output difference between queries with and without taking the prior knowledge into account. Over the iteration, we mainly play with the dummy conditions combinations we are using. Once this list of differences generation is done, we use statistics to compute the probability of whether s=0 or not.

By principle, the cloning attack requires weaker conditions for the first step. It also makes less assumptions. This makes it more effective when working on large scale datasets. However, the generation of distinct and precise dummy conditions is a key element for the attack to succeed. And this part in particular can be pretty much complicated to handle.

We can once again notice the attack uses Diffix's mechanism against it. The sticky depends on the condition clause within the query on one hand, and on the raw returned result on the other hand. The attack plays smartly with the condition clause using the dummy conditions without affecting the user set returned. This will obviously affect the status noise which depends exclusively on the condition clause, but not the seed used for the dynamic noise. This opens the way to some averaging attacks which can cancel the static component of the noise.

Chapter II – GDA Score

2.1. Introduction

The General Data Anonymisation (GDA) Score is software developed by the Max Planck Institute for Software Systems and Aircloak GmbH. Its main purpose is to propose a framework to measured different de-identification techniques and compare them. This score computed by the software measures essentially defense and utility. The project also introduces an API for attack development and score measure.

The software is still in its early stages of development. The first working version has been introduced early 2019. Until now, only basic features are implemented. Many others remain lacking. The API is also still full of bugs.

Comparaison with ARX

ARX implements different de-identification technologies. It takes a raw dataset as an input and generates an anonymised one. GDA Score however doesn't interfere with the anonymisation process at all. Its role is rather to examine the distance between the these two datasets, raw and de-identified, in order to assess this technology's defence and utility performances. So while ARX takes care of the anonymisation process, GDA Score comes behind to assess it and eventually identify tracks to improve it. In a way, both software can be seen as complementary.

2.2. Set up

The code source was cloned from the git repository hosted on *https://github.com/gda-score/code.git*.

There are four environment variables theoretically used by gda-score:

- GDA_SCORE_DIFFIX_USER
- GDA_SCORE_DIFFIX_PASS
- GDA_SCORE_RAW_USER
- GDA_SCORE_RAW_PASS

These environment variables are fundamental to connect to the databases server, and so run attacks. Default credentials for these variables are present in the configuration file. But to get credentials properly, the developer team precise in their documentation that we should ask it from *contact@gda-score.org*. That was the get-in-touch with Paul Francis and his assistant. The first attack tests failed though. After some mail exchanges with M. Forouzesh, it was precised that the code is being developed on Windows OS, and compatibility issues with Linux-like systems are still to be addressed. The problem was later solved, when it appeared that on Linux systems the environment variables aren't even necessary for the software's functioning. What is to bear in mind however is that the attack must be executed from the command line from the folder code/ as a working directory.

2.3. Technical description

GDA Score comes with a Command Line Interface (CLI) for set up and configuration. The software is actually being developed on Windows. Therefore, many compatibility issues with Linux are still to be handled.

The software also comes with an API to develop attacks, which allows the software to measure defence of an anonymisation technology. These attacks are actually implemented independently from the software by

any developer. The software then executes them, in accordance with a user-specified configuration, and generates defense scores to assess how good the attacked mechanism ensures data privacy.

2.3.1. Architecture

The gda-score API is based on three main classes: *gdaAttack*, *gdaScore* and *gdaTools*. The developers precise in their documentation that, indeed, the project is still in its early stages of development. The whole code is fragile and can fail.

The project is divided into 3 main sub-modules:

- gdaAttack: functions to perform attacks on an anonymised data set.
- gdaScore: evaluates an attack and generates utility and defence scores.
- gdaTools: this module regroups some utility functions, among which setupGdaAttackParameters. SetupGdaAttackParameters() is used to setup the parameters for the attack such as anonymisation method used and dataset to attack (most required).

2.3.2. Anonymisation mechanisms

The gda-score API comes already defined de-identification technologies.
It's important to note that for pseudonymised tables, some columns can be suppressed.

2.3.3. Documentation

The documentation was generated using pdoc3. If not done, the library can be installed using pip3. Dependency packages "pyinquirer" and "pyfiglet" were required to be installed using pip3.

The run the documentation generation, we run the following command from the folder code/:

```
pdoc -o ./docs --force --html --template-dir ./docs/template --filter gdaScore,gdaAttack,gdaTools gdascore
```

2.3.4. Databases

The GDA Score API comes with a number of databases hosted on db001.gda-score.org. These databases are:

- Banking: comes with three attackable tables, which are "accounts", "loans" and "transactions".
- Taxi: comes with one attackable table "rides".
- Census: comes with one attackable table "persons".
- Scihub: comes with one attackable table "downloads".

For the moment, this database host is hard-coded in the API's source code, and databases can be added only manually by the developers team.

2.4. Score measuring

One of the main objectives of the GDA Score project, as its name suggests, is to give scores to different anonymisation schemes. By measuring de-identified technologies, the software provides a framework for comparison of different anonymisation approaches.

De-identification technologies seek to protect individual's data privacy in a dataset, while keeping a minimum level of utility and value for data analysts. The stronger the algorithm's defence is, the less valuable it is, and vice-versa. Finding an acceptable balance between these two notions is one of the difficulty axis in anonymisation. Therefore, two different types of score measures are available in the software: defence and utility. The first one measures the strength of a de-identification algorithm against a specific attack. The second measures the analytical value of a de-identified dataset. The two measures are based on query-

based approaches. Nevertheless, they are completely independent. The way defense measure is computed has absolutely no ties to how utility is computed.

2.4.1. Defence

The defence score measures how strong a de-identification technology is. The lower the score is, the stronger the algorithm is. The strength of the algorithm is measured based on the attack which was carried out against it, the anonymity criteria we are selecting and the anonymised database which we are attacking.

The defence score is basically calculated by compiling five different sub scores at once: susceptibility, prior knowledge, claim probability, work and confidence improvement. Susceptibility represents the amount of data which can be attacked regarding the whole dataset. Prior knowledge measures the anterior knowledge of the dataset the attacker acquired outside the framework of the attack, from an open source documentation for example. It concerns information available in the de-anonymised raw database, but not available in the de-identified one. The claim probability represents the accuracy of the claims made by the attacker by the end of the attack workflow (cf. Making claims). The work corresponds to the amount of effort needed to conduct the attack. It is mainly based on the number of queries executed. The confidence improvement is relative to the amount of attacked data regarding the total amount of attackable data. It represents how confident and audacious the attacker is while de-anonymising data.

The anonymity criteria used in the GDA Score software are based on the “EU Article 29 Data Protection Working Party Opinion on Anonymisation”. The three available criteria for the moment are: singling-out, inference and linkability. Singlilng-out characterises the ability to uniquely de-anonymise some or all records in a dataset. Inference is the process of deducing the of one particular attribute giving the value of some or all other attributes. We usually try to infer the value of a sensitive attribute. And linkability is the possibility to hold a tie between two records describing the same particular individual. These records can either be in two different tables of the same database, or in even from two different databases. Anyway, linkability tests require two datasets, one protected by the de-identification scheme and the other is public.

The defence score also rely on the attack carried out against the de-identification mechanism. The different five defence-related scores are actually produced by running the attack. For each anonymisation scheme, we usually have many different attacks possible. As some attacks can be more efficient than others, the score would also be different. In those cases, we usually tend to analyse the worst case, which corresponds to the most lethal attack at our disposal. The score can also evolve with the evolution of the attacks. The more attacks are developed, the more weaknesses in these mechanisms are discovered. The developer team from the MPI-SW are “expecting that overtime, more attacks will be implemented and the GDA Score will better represent the true protection offered by the de-identification scheme”.

2.4.2. Utility

The utility score measure how valuable an anonymised dataset is. The higher the score is, the more valuable the dataset is. Carrying out a de-identification on a database will inevitably cause either (i) the suppression of some attributes, and therefore the loss of information, or (ii) the alteration and changes on some sensitive values, and so a distortion of information. In many cases, we have both loss and distortion of information. To measure the utility, the software actually conduct two different and independent measures: coverage to determine the amount of lost data, and accuracy to see how accurate the remaining data is regarding the original raw dataset.

2.5. Attacks

2.5.1. Introduction

One of the best features offered by the GDA software is the possibility to implement attacks and test them. These attacks are implemented in independent files, using the GDA Score API coded in Python3. The implementation needs to follow some rules relative to the API though.

2.5.2. Attack's implementation workflow

The GDA Score API provides a software framework to implement and execute attacks. An attack implementation however needs to follow a certain number of rules and logical steps, some are mandatory and others are optional.

Configuration:

First of all, we define the configuration of the attack we wish to carry out. The configuration can be set up in an independent JSON file or within the code itself as a Python3 dictionary in a format which is also very similar to the JSON format.

Within the configuration, we first define the attack's type and criteria. The criteria must be one of the three EU Article 29 criteria for anonymity: singling-out, inference and linkability. Then, we select the table we are targeting. We thus precise a database and a table to the software, given that this database is among those hosted on the software's server. Last, which the most important, we need to select the anonymisation type we are attacking. As mentioned before, different varieties of the database are hosted on the server, each obtained after a particular anonymisation operation. Selecting the anonymisation type would specify to the program which version of the database is to be targeted. The anonymisation types available so far are:

- *No Anonymisation:* which corresponds to the de-anonymised raw dataset
- *Pseudonymisation*
- *Diffix v19.1.0:* As explained previously, Diffix acts more as an SQL proxy server. Therefore, the algorithm is actually implemented on the server directly. The PostgreSql interface takes on its charge the task to process the query (fetching results, adding noise ...) and return the adequate answer.
- *K-anonymisation:* two models are available, for k=2 and k=5
- *Pseudonymisation*: with column suppression

Prepare the attack:

Once the configuration is defined, it will be processed by some utility functions, in order format the necessary parameters to pass on onto the Attack class.
An attack is necessarily conducted upon a *gdaAttack* class, which takes the previously formatted parameters as an argument. The *gdaAttack* class provides a large set of attributes and methods to prepare and conduct the attack. After the instantiation of the attack, we use different class methods to gather information such as the name of the table and columns. We can have a column list for both raw and anonymised database. We are also able to retrieve the possible values for each attribute and its number of instances. We rarely need all these information altogether. Depending on the attack, we request only the information we need.

For some attacks, we might need a whole set of records, whether filtered with some conditions or not. For this, we use the methods *askExplore()/getExplore()*. The latter requires some information as argument, containing the SQL query to run and the targeted database, whether the raw or the anonymised one. And that's exactly why we use these methods, because it allows running SQL queries on both types of databases. It's not the case for *askAttack()/getAttack()* methods for example. The argument is passed in the format of a dictionary.

Once we get our information, we can process them into some algorithm if needed. This would depend on the carried attack as well. By the end, we should be able to generate a set of SQL queries, which are going to fetch some very targeted information.

Launching the attack:

The generated SQL queries set is run using the methods *askAttack()/getAttack()*. We are using these methods, because we suppose that we reached here the point where we would only trigger the anonymised database. All queries' effort will be taken into consideration while computing the required work for the attack, and so influence the ultimate score. As explained previously, the work score is one among five scores which average will produce the final defence score. One should always pay attention to get the complete expected answer. Some bugs or unexpected behaviours might pop up. The best practice is to set regular controls and checks while coding, also set output points to control the workflow in general.

One must always keep in mind that the ultimate point of launching the attack is to open up the way for making claims. The retieved data from the attack queries usually require further treatment and processing, some might involve attack-related algorithms.

Making claims:

All the knowledge acquired from the previous steps, whether from the attack preparation or the attack queries results, is to serve this ultimate step in the attack workflow. The attack is aimed at de-anonymising one particular or multiple records from the targeted table. Each of the set of claims is the description of a record. A claim can be either a guess or a piece of knowledge. In both cases, it's going to be a statement, a concatenation of attributes' values which represents an eventual record of the raw database. The claim can also be set to either 'True' or 'False', which means that we can claim that such a record with the specified values indeed exists in the dataset, or that no similar record is actually present in dataset.

A guess formatting might differs depending on the anonymity criteria we are trying to challenge. But all three criteria share common points regarding it. Regardless of the criteria, we would for each single claim concatenate some or all attributes' values belonging to a particular record. This guessed record isn't actually present or accessible as it is in the anonymised table. The software will in fact check that out, and verify that such a record with these guessed values exists in the raw table.

One should however pay attention to include the known records, most notably those which where retrieved directly from the raw database while preparing the attack. This is important because these data will count for the prior knowledge score. As explained previously, the prior knowledge score is one among five scores which average will produce the final defence score. Prior knowledge can also refer to other records or values the attacker has got outside the attack program. During my work on the GDA score software, I didn't encounter any similar use case. But it's possible to occur, especially with the development of the software and the introduction of new features.

Getting results:

After our attack workflow is over, wa can finally check the results. The *gdaAttack* class provides a method to retrieve results and different detail scores. We can then define a *gdaScore* class to compute the final scores. This class's constructor will take the previously retrieved results as an argument. The class provides methods to compute the final score. A JSON file will be then generated, detailing all configuration parameters, attack results and computed scores. This file can be then used to have a graphical display of different charts.

Before quitting the program, a good practice is to clean up the cache and exit the program properly. Responses for executed SQL queries, databases, status of the current attack and many other data are cached by the GDA score API to ensure its functioning. These cached data might interfere with other future attacks. That's why it's better to quit the program properly.

2.5.3. Attack's execution

During the internship, the main system used is a Linux-based. As mentioned previously, some troubleshooting might be required to execute the attack due to compatibility limitations. An Internet connection is required, the attack file to execute must be moved to the depository's root folder *code/*.

Chapter III – Contribution and attacks development

3.1. Overview

3.1.1 Description of already implemented attacks

The developers team already provides some attack examples on Github. These attacks are basic and don't really attack anything. They serve as example to put into evidence the implementation workflow from a programming point of view.

3.1.2. Implemented attacks

After studying the attack files provided by the developers team, I passed on implementing my own attacks. First, some test attacks were developed. The main point was to better apprehend the gda-score API. Then came the implementation of the noise-exploitation attack porposed by Imperial College Londond. The attack already figures on the to-do list on the GDA Score website. Later tests weren't very much successful. This was the first attempt for contribution to the GDA Score project. The second attempt was the coding of a distance-based attack. This attack was successful and gave the expected results.

3.2. Test attacks

I first developed three slightly different attacks to better apprehend the GDA Score API and deeper practice the software. Each attack was implemented on an independent file, and run separately. The main idea of the attacks was to look for values in databases with a very low number of instances. Then gather information about records showing these instances and try to single out some de-identified records.

First and second attacks are two versions of almost the same attack, with few differences although. Their information gathering phase share a common side; they are limited to gathering information from the targeted only. The third attack is slightly more sophisticated on this aspect. The program would step over the targeted table's border, and seek information from other tables. It looks foremost for common attributes between tables, which are concerned with these sought rare values.

3.2.1. First test attack

This assumes a sensitive attribute and a particular targeted value v for this attribute. It aims at singling out some specific users with this value v. The algorithm basically operates a standard search for records fulfilling this requirement.

3.2.2. Second test attack

The attack aims at singling out individuals with specific characteristics. The idea is actually to look for values with very low number of appearances, regardless of the attribute. We refer to these values as "interesting values". The algorithm will after try to retrieve all records with one or more of these interesting values. Then we try to find collisions between these records. This attack is the very basic threat intended to be prevented by the k-anonymity model.

3.2.3. Third test attack

This attack is similar to the second, and should be considered rather as its extension. Instead of looking for these "interesting values" just in the targeted table, we actually try to extend our search to the whole database. The algorithm also focuses its search on tables with shared columns. Thus, it tries whether if a

record was identified in a table, and this table shares one or more column with another table, then might learn more about this individual. The ultimate purpose remains singling out this user.

3.2.4. Tests and results

Despite the algorithm's idea being focused on singling-out records, the attacks were tested on all three anonymity criteria. That was foremost to observe how the software will behave according to each criteria, and so learn more about this side the framework.

Tests on the raw datasets were very successful with a de-anonymisation rate of 100% each time. Tests on de-identified data, for most of the de-identification technologies, weren't however as successful. The de-anonymisation rates were almost all null.

3.2.5. Analysis

The success of the scripts regarding raw databases was a reassuring indicator that the implemented attack workflow and methodology were correct and working. The attack was actually trying to de-anonymise already de-anonymised data, which explains its obvious success.

The following failures however are quite expected. It's for example the very purpose of k-anonymity to avoid having easily remarkable values. It's thus normal that the attack failed, as it tries to challenge the de-identification technology over its home pitch and main strength point. But it's interesting to notice how exposed data is on the raw dataset. Whereas from the moment we add some anonymisation, we immediately see the impact on the defence score.

3.3. Noise-exploitation attack

3.3.1. Implementation

All different varieties of Diffix's noise-exploitation attacks were implemented: differential attack, cloning attack and greedy cloning attack.

The programming phase of the attack was quite long. The attack's paper already provides some pseudo-algorithms for its implementation. The programming of these pseudo-codes is much more sophisticated than it might seem at first. Other components and sub-components had to be implemented as well from scratch.

3.3.2. Tests and results

It's important to notice that a single test for any version of the attacks, differential or cloning, might take up to an hour on a standard users computer. A set of tests was carried out. In order to save time, these attacks are usually executed overnight. Specific automation codes were written to sort it out. Some issues regarding the server hosting targeted datasets occurred during these tests. It happened several times that during the tests, the server goes down. Few minutes later, it's usually up again. But it interrupt some scripts however, which implies even more time loss.

None of the executed tests gave a positive outcome. All tests failed, the outcome is almost always pretty much the same: the dataset is non-attackable. This means that the program don't even make it to the attack's algorithm, but it actually blocks in the selection phase.

3.3.3. Analysis and critics

Although in the beginning it seemed surprising, it's actually expected that the attack fails.

After the publishing of the attack by CPG-ICL, Paul Francis, director at the MPI-SWS published a blog post regarding the attack. He claim that the attack was unpractical in real cases. He said that "the conditions under which it could work are so rare as to be practically non-existent". He actually conducted empirical analysis on the tested datasets. After which, he concluded that that the attack would only sometimes succeed depending on the targeted datasets.

A first hypothesis to the failure of the attack is then, that the tested databases are non-attackable, which is the same output returned by the program. After the failure of the attack of the embedded datasets in gda-score, I contacted Paul Francis to discuss the possibility of adding the datasets tested by CPG-ICL. The adding wasn't possible, but he mentioned however in his reply his doubts about the attack's efficiency. He stated in his mail: "In my measurements a couple years ago, I found that only something like 1 in 100K values were attackable".

The CPG-ICL from their side declared in a blog post, on 17. May 2018, that they would evaluate the analysis made by Paul Francis, but published nothing about it since then. I asked with Pr. Nguyen to have a copy of their source code for comparison and analysis. We also asked for their position about claims concerning the practical inefficiency of the attack. Until the end of the internship, they gave no positive reply. Yves-Alexandre de Montjoye, head of the team, precised however that they were willing to make all their work open source. But legal issues prevented them from publishing.

3.4. Distance-based attack

3.4.1. Introduction

The attack was at first developed during an inter-university contest held in INSA Lyon by a team of students from INSA Centre Val de Loire. The same team continued developing their approach within the context of their computer security project. The two team members responsible for the attack-related task are Théotim Dejean and Stéphane Laserre.

3.4.2. Description

The main idea of the attack is to compute the distance between a de-identified record and a raw record, or a portion of it. The attack assumes that we dispose of a certain prior knowledge of the raw database. More specifically, for attributes present in the anonymised database, we assume that we dispose of their raw values for each record. This prior knowledge can be fetched from public and open source for example. We have thus, records of both raw and de-identified databases, but usually limited to attributes present in the de-identified database. Practically, the amount of raw data is usually less than the anonymised one for both, the number of known values and number of records.

The distance between two records is the average distance between their respective values for each common attribute. The distance between two values of an attribute depends on the attribute's type. For example, for integers or floats, we can consider that the distance is the absolute value of the difference. For binary attributes such as 'Gender' ('Male' or 'Female'), we consider that the distance is either 0 for identical values, or 1 when different.

The attack algorithm will actually go through all de-identified records, and for each, try to compute the distance between this record and every portion of raw record we have. We actually try to find out the closest raw record in term of probability, corresponding to the smallest distance, to the selected de-identifed record. We then make an inventory of all matches made, and so our guesses.

An further optional development of the attack, would also establish a similar match table of records, but for records which are most unlikely to be the same. We would then match these records by selecting the longest distance instead of the smallest one. During the claim phase, these matches can be added to other guesses, while setting the guess label to False.

3.4.3. Implementation

Three versions of the code were developed. The first was biased due to an algorithmic error. It didn't concatenate the consecutive single guesses as foreseen. The second version addressed this issue, and the code's behaviour was made more sense. But for these two first implementation versions would only consider matches based on the smallest distance. We are therefore simply looking to de-anonymise records. A third version supported the extra option which consists in making false guesses, while being labelled so, for very far distanced records.

3.4.4. Tests and results

Tests were conducted while configuring different parameters. Each time, we would vary one parameter with the other parameters fixed. Different test families were then carried out:

- Code versions
- Anonymity criteria
- De-identification schemes
- Different databases

Code versions:

The first version tests failed. With control outputs and debugging, it left no doubt that the algorithmic approach in sorting guesses was wrong. This version was thus abandoned.

The main adopted version is the second one. The tests are working fine and the results are very acceptable. The script's behaviour was quite expected, and not many errors were reported. The version was used as a fixed parameter for following families of tests.

The third version was considered at the beginning, but the false claims were troubling the results and causing several bugs.

Anonymity criteria:

Singling-out criteria is the main relied on criteria. It works just fine, as the attack's aim is to uniquely identify records. It's also used as a fixed parameter for the other test families.

Inference criterion was tested, but the tests were a failure. This was expected as the attack algorithm isn't trying to infer any particular value.

Linkability criterion wasn't even given a shot. In no way, the attack is involving two different datasets, nor tying them up.

De-identification schemes:

Four different de-identification technologies from the option list GDA Score offers were successfully tested: the k-anonymisation for both k=2 and k=5, and the k-pseudonymisation also for k=2 and k=5. The k-anonymisation, with k=2, is used as a fixed parameter for carrying out other test families.

The non-anonymisation worked partially. The algorithms bugs during the claims stage. The Diffix v19.1.0 technology is returning an error of 0 anonymised user fetched, and so its inability to carry the attack further on.

Different dabases:

Tests were successfully on three different datasets: 'loans' and 'accounts' tables from the Banking database, and the 'rides' table the Taxi database. The 'accounts' table of the Banking database is used as a fixed parameter for the other test families.

The 'persons' table from the Census database and the 'downloads' table from the Scihub database returned weird errors. They indicate that during the attack preparation stage, 0 plain user and 0 anonymised user were retrieved. Therefore, the attack couldn't advance further. The 'transactions' table from the Banking database also returned an error.

3.4.5. Analysis

Successes and failures regarding de-identification technologies were almost all very much expected, except for the raw datasets. The idea of the algorithm is intended to essentially challenge pseudonymisation and k-anonymity. And that worked.

The implemented distance-based attack is subject to a margin of improvement. The distance computing algorithm still doesn't treat many cases, such as date time values or formatted decimal numbers.

Chapter IV – ARX software analysis

4.1. Introduction

4.1.1. Software overview

ARX De-Identifier is a GUI software used in data anonymisation. It takes raw dataset as an input and return the de-identified dataset. It implements features which allow the user to configure the anonymisation process in order to set privacy parameters such as k-anonymity and l-diversity. Moreover, once the dataset is anonymised, the software provides other features for utility and risk analysis.

4.1.2. Team and use cases

The team behind ARX are actually researchers affiliated with the Technical University of Munich (TUM) and the Berlin Institute for Health (BIH). In the beginning, the purpose behind developing ARX was actually to provide an fast and efficient way for biomedical data anonymisation.

4.1.3. Contribution

The software is made of 75873 lines of core code and 68758 for the GUI. This makes it a pretty heavy code to apprehend when some editing is needed.

When adding a new feature to the software, the developer in charge is asked to follow ARX team “coding style and minimise changes in the existing code”. Including tests to outline the brought changes is also required. Any developer willing to take part in the software coding has to abide by a “Code of Conduct”, which is rather behavioural than technical.

4.2. Architecture

4.2.1. Generic components

The ARX project provides a public API which represents the very core the ARX software. It can be see as the application layer, ie. the main program and other functional features. The main purpose of having the ARX API completely independent is so it can be reused within other software or frameworks. The ARX code covers, in addition to the software’s main code, the GUI code, a set of test codes and example files for the ARX API manipulation. All of those are independent and developed separately. The GUI code can be considered as a sort of a presentation layer. Examples are also available to better explain how to use the API.

4.2.2. ARX’s API architecture

The ARX API is composed of different packages, each represents a component of its own role.

- Core classes represent the main interface for the ARX software. They include the software’s launch code and overall configuration, and are in charge of data and result synthesis from other components.

- Hierarchy generation regroup classes which implements models of data mapping. These classes enables building hierarchies for categorical and non-categorical values, and mapping data in a form that can be handled by other software components.

- Data specification classes define different used types of data and their attributes. They also offer features for interaction with data, such as dataset importation and sorting.

- Privacy models component gather a set of classes implementing several privacy criteria. Each class represents a criteria, and all classes are inherited, directly or indirectly, from PrivacyCriterion class.

- Risk analysis-related classes implement a number of risk calculation model in order to give estimations for de-anonymisation risks.

- Data utility metrics component provides measures for dataset utility after de-identification. The related classes implement different variables to be computed (accuracy, loss ...) in order to provide the utility measure.

- Utility analysis provides classes and functions for statistics calculations on de-identified datasets.

- Solution space-related classes implement needed features for the de-identification process. The ARXLattice class

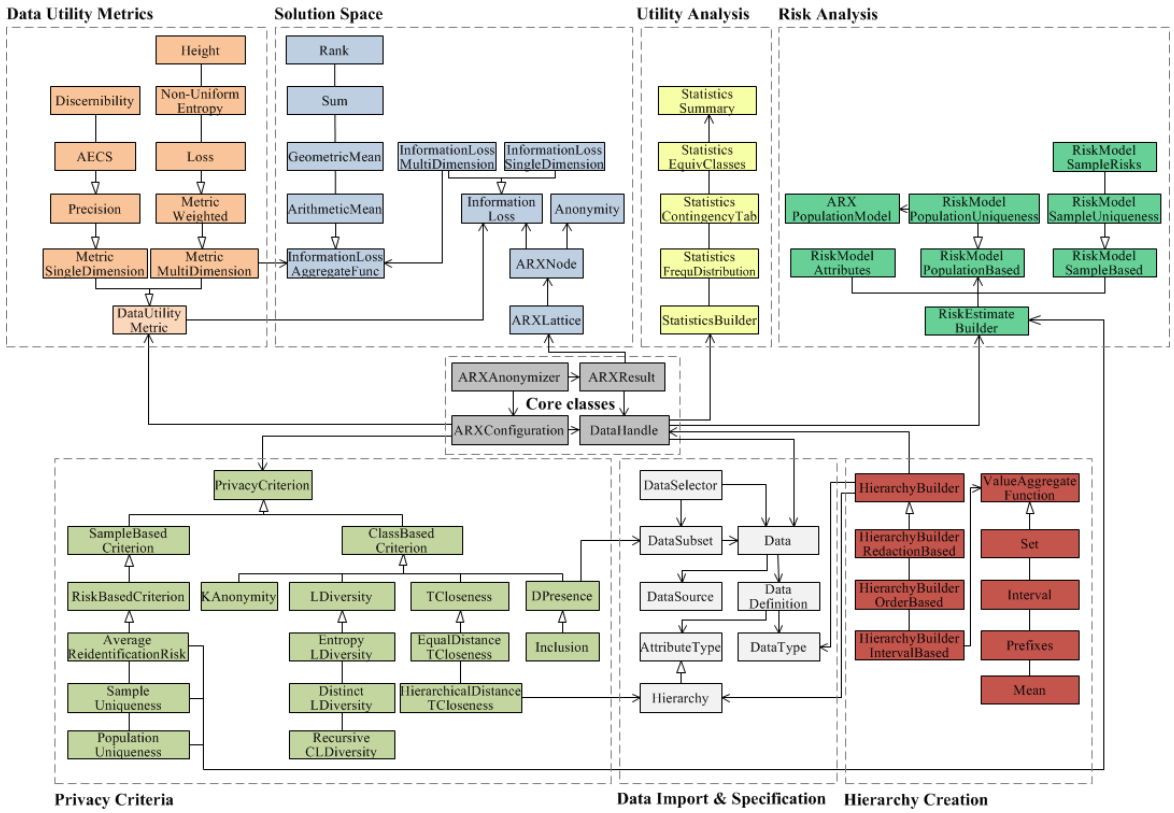

Figure 4: ARX's API architecture

4.3. Development

This section mainly describes the implementation of a privacy criterion (k-anonymity), in order to give a clue for adding further criteria. In the contribution's page on ARX's website, "adding further privacy criteria" is labelled as a task of "Very high difficulty". The following might explain the reason.

We start by cloning ARX's source code from https://github.com/arx-deidentifier/arx.git. We consider afterwards arx/src/main/org/deidentifier/arx/ as root folder.

If I start from the tail of the chain, then I'll begin with the KAnonymity class located in criteria/. This class implements just few charasterists of the criterion, such as the model's parameters. It is however inherited from ImplicitPrivacyCriterion class, which is itself inherited from PrivacyCriterion class. The two last criteria are located in the same folder as KAnonymity, and they implement much the code, especially for PrivacyCriterion.

In the PrivacyCriterion class, we can notice that it's within that the data is mapped and sorted. To do so, we use core classes such as ARXPopulationModel and data specification classes such as DataSubset. Both latter classes are located in the root folder. At this point, we start to notice how different classes are interconnected and involved in each other. The DataSubset class is important to apprehend, because it describes the schema of the data on which we are going to manipulate in a programming operation. But this class itself is interconnected with other classes, for which some editing is necessary to take into consideration the new criterion to be.

Now If I start from the head of the chain, I'll begin by looking at the ARXAnonymizer first. This class is the very first class to call when starting the software, or at least the part which interesting for this task. The ARXAnonymizer class requires a configurartion implemented by ARXConfiguration, uses ARXLattice actively, and among other things, implements a solution space, to be filled for any eventual de-identification process. The solution space is implemented by the class SolutionSpace, located in framework/lattice. This class is crucial; because it describes the mapping of the data we should come to if to code a new criterion.
Just like for the other direction, this class itself is interconnected with other classes.

Introducing a new criterion, requires also editing risk and utility analysis related classes, such as RiskEstimateBuilder and StatisticsFrequencyDistribution. These two last classes are respectively located in risk/ and aggregates/. Risk and utility classes require slight editing because they need to take into consideration the new criterion, which eventually is going to be measured. The editing is light because after all the software will just take the output solution (de-identified dataset) and apply its algorithms on it.

The main purpose of the task was to re-implement the k-anonymity criterion in ARX, in order to define the workflow needed to implement any other criterion. The ultimate goal is to later try to implement ki-anonymity criterion. Unfortunately, the task didn't work out. The source code to study turned to be far longer than expected, and on a high level of sophistication. Besides, the remaining time was too short. I tried nevertheless to identify some portions of the code which are affected.

I would to highlight that this description might be partially false. Actually, along the code study, I always learn new things which turn later to be false. My work on the code, being unfinished, might reveal some mistakes.

Conclusion

The internship took place in three main stages. The first was dedicated for self-learning, documentation and reading papers about data privacy. During the second stage, I started the core work upon gda-score, while still deepening my knowledge in data anonymisation. The third was a switch to working on the software ARX De-identifier.

The first part was quite short, yet very determining. I started the internship with almost no background knowledge about data privacy as a field, except for some very surface material I read before. I had therefore a lot to catch up for the courses at first, in order to better understand different issues and concepts of this discipline. In a second time, I had to acquire more specific knowledge about the material and tools I am going to work on. This concerns more documentation about gda-score, Diffix as an anonymisation mechanism, generic attacks … While getting more and more involved in the discipline, it's during this period that I started liking it. This was very important to ensure for myself good working conditions for the rest of the internship.

The second part is by far the longest compared to the two others. I started by playing with gda-score, see how it works and discovering it. Quickly, I started testing things and implementing attacks. The first attacks were basic, just to get more into it. Meanwhile, that's when I first got in touch with MPI-SWS researchers, led by Paul Francis. Then more serious tasks came up with the implementation of the noise-exploitation attack. The attack was quite sophisticated and required deep understanding of the paper introducing it, and also the Diffix anonymisation mechanism. Besides, there imprecisions and some controversial aspects of the thing which were to be put into context, understood and taken into consideration. The programming phase was also long due to the sophistication of the algorithm, as were the tests too. Despite the disappointing results, Ivery much enjoyed working on that part. I was learning new stuff in data privacy, apprehending a specific research paper, programming and analysing information, all in the same time. Regardless of the cause, it's a pity that the tests didn't make it through. Working on this attack also made me handle some basic logistical issues which are likely to show up in those kind of working environment. The test phase was related to the GDA Score framework from one side, and to the researchers who developed the attack on the other side. This made difficult to conduct smoothly. The work on gda-score hasn't been concluded without any significant success. After the tests for the noise-exploitation attack were put on hold, I moved to implement the distance-based attack. This attack had more successful results than its predecessor. It was a major breakthrough in the gda-score phase as I was finally disposing of satisfying results for defence measures. Working on Diffix's sticky noise is also interesting. It had me practicing some analytical skills, which are crucial to constantly keep improving.

The third and final part corresponds to my work on ARX. This part was without any doubt the most difficult, and unfortunately the least productive. I had as task to re-implement some specific feature of the software. Thus, I had to study the software's source code first, in order to assess the feasibility of the task. I made a mistake at this by underestimating the difficulty of the job. The code to study was very heavy, making tens of thousands of code lines. Besides, the software was subdivided in different components, and I quickly made the statement that only one or two are needed to work on for the task. Later on, I discovered how interconnected all components are between each other. And that was a major failure. Nevertheless, I tried to provide some clues about how to implement privacy criterion in ARX, in order to limit the damage. This might save some time for further interns or any developer who wishes to carry it further on.

In parallel with the scientific side, the fact that this is a research internship was very beneficial academically. As a research intern, I had to work on a research project, study the state of the art of a scientific branch, read and apprehend scientific papers, interact with other researchers and follow some very specific methodologies for the research job. There are also several issues one might have to deal with when working as a researcher. All of this made me understand much better the researcher job. By having a research internship, my intention since the very beginning to actually discover all these aspects. It goes with my short term study perspectives of pursuing a research master, and eventually a PhD. Moreover, it helps me very much for my professional plans.

Bibliography

[1] Article 29 Data Protection Working Party Opinion 05/2014 on Anonymisation Tecniques. http://ec.europa.eu/justice/data-protection/article-29/documentation/opinion-recommendation/_les/2014/wp216 en.pdf

[2] Paul Francis, Sebastian Probst Eide, and Reinhard Munz. Diffix: High-Utility database anonymization. In Privacy Technologies and Policy, pages 141–158. Springer International Publishing, 2017.

[3] P. Francis, S. Probst-Eide, P. Obrok, C. Berneanu, S. Juric, and R. Munz. Extended Diffix. ArXiv e-prints, June 2018.

[4] Gadotti A., Houssiau F., Rocher L., Livshits B., de Montjoye Y. A. (2019) *When the signal is in the noise: Exploiting Diffix's Sticky Noise*. 28th USENIX Security Symposium (USENIX Security 19).

[5] Fabian Prasser, Johanna Eicher, Helmut Spengler, Raffael Bild, Klaus A. Kuhn, **Flexible Data Anonymization Using ARX — Current Status and Challenges Ahead.** J Software Pract Exper 50, 7 (2020);1277-1304

[6] Fabian Prasser*, Florian Kohlmayer*, Ronald Lautenschlaeger, Klaus A. Kuhn. **ARX – A Comprehensive Tool for Anonymizing Biomedical Data.** Proceedings of the AMIA 2014 Annual Symposium, November 2014, Washington D.C., USA.

[7] Florian Kohlmayer*, Fabian Prasser*, Claudia Eckert, Alfons Kemper, Klaus. A. Kuhn. **Flash: Efficient, Stable and Optimal K-Anonymity.** Proceedings of the 4th IEEE International Conference on Information Privacy, Security, Risk and Trust (PASSAT), September 3 – 5, 2012, Amsterdam, Netherlands.

[8] Florian Kohlmayer*, Fabian Prasser*, Claudia Eckert, Alfons Kemper and Klaus A. Kuhn. **Highly Efficient Optimal K-Anonymity For Biomedical Datasets.** Proceedings of the 25th IEEE International Symposium on Computer-Based Medical Systems (CBMS), June 2012.

[9] Benjamin Nguyen, Claude Castelluccia, Techniques d'anonymisation tabulaire : Concepts et mise en oeuvre, Bulletin 1024, 15:23-41, 2020

[10] Axel Michel, Benjamin Nguyen, Philippe Pucheral, The Case for Personalized Anonymization of Database Query Results, DATA (Revised Selected Papers) 2017, J. Felipe et al. eds., in Communications in Computer and Information Science, 814, Springer, ISBN 978-3-319-94808-9, pp 261-285, 2018.

[11] https://www.gda-score.org/, 28/08/2020

[12] https://pypi.org/project/gda-score-code/, 28/08/2020

[13] https://cpg.doc.ic.ac.uk/blog/aircloak-diffix-signal-is-in-the-noise/, 28/08/2020

[14] https://aircloak.com/statement-regarding-the-attack-on-diffix-by-imperial-college-scientists/, 28/08/2020

[15] https://aircloak.com/report-on-the-diffix-vulnerability-announced-by-imperial-college-london-and-cu-louvain/, 28/08/2020