Design and Use of PreTPost Framework

Hamid Ebadi

Department of Computer Science and Engineering,
Chalmers University of Technology
Gothenburg, Sweden
hamide@chalmers.se

Abstract. The PreTPost [3] is a generic local differential privacy framework with the aim of verifiably respecting the privacy expectations of individual users that are subject to data collection. This paper describes the design and use of PreTPost framework for performing differential privacy analyses in the local setting. Using a real world scenario, the construction of an analysis in the curator side and its execution in the user side is studied. As the analysis is constructed and executed, different aspects of the framework are illustrated. Furthermore, we discuss the purpose of some important design decisions and their implementation detail.

1 Introduction

The *PreTPost framework* [3] is designed for performing statistical analyses while respecting the privacy expectation of users. In the framework, the privacy risk caused by execution of an analysis is objectively quantified using differential privacy. In differential privacy, the value that individuals possess is protected by accepting a limited amount of uncertainty or noise in the outcome. The differential privacy guarantee is achievable when results from an analysis are similar regardless of the data provided by any individual. From the analyst's perspective, the similarity prohibits firm decision making based on the data contributed by an individual, but it is acceptable for statistical inference and overall decision making.

The framework is implemented in Python but does not use any special features from the language and can as well be implemented in other languages. Python is chosen for its clarity and its portability to variety of hardware architectures but as we learn (Section 5) even binary executables are acceptable as a component of queries in the framework.

In this paper we are not aiming to repeat the fundamentals of differential privacy and the PreTPost that are discussed in [3, 1], but instead we look at the system structure and architecture from practical perspectives that are missing in the reference paper. We particularly look at the framework from the perspective of analysts, who design the analyses, and

the system architecture, who builds and verifies the user side of the system. Through a series of scenario notes, we use a concrete example of an Internet Service Provider (ISP) that is interested to study users' router configuration.

There are two flavours of differential privacy, the local and the centralised model. In the centralised model users trust the curator with their data and they apply differential privacy correctly; on the other hand, the users are in charge of protecting their own privacy in the local model. The PreTPost is a local framework that permits users to verify the consistency of the queries that are sent by the curator with the user's preference and expectation. In Section 2 we look at the communication between a curator and users. We learn how users and the curator negotiate and agree on their particular privacy preferences and parameters.

The PreTPost can use variety of mechanisms to achieve differential privacy, but in the example used in the tutorial we mainly focus on generalisation of one classic mechanism known as *randomised response* that is shown to be a basic idea behind many other algorithms. To illustrate a simple example of this schema, assume that participants in a survey are instructed to toss their own two fair coins before answering a Yes/No question. If both coins come head users respond "yes", if both come tail users respond "No" regardless of the actual answer; users otherwise answer truthfully. The plausible deniability in the schema permits individuals to deny their responses by claiming that a response is the outcome of a coin toss and therefore not truth-worthy, while at the same time these noisy data is meaningful when studied in aggregation. In this case, the analyst who is aware that $\frac{1}{4}$ of the responses are incorrectly broadcasted "Yes" and $\frac{1}{4}$ are incorrectly broadcasted "No", gets a rough frequency estimate of the answers. This method is shown to provide $\ln(3)$ differential privacy but generalisation of this schema to arbitrary ϵ value is possible using biased coins[1]. In Section 3, we see an example of these components and how the curator encodes the randomised response algorithm in a query.

Similar to other differential privacy frameworks, the desirable level of privacy is specified with *budget* value ($\mathcal{E}$). On the other side, the maximum possible risk[2] that release of results causes on an individual is specified with small epsilon (ϵ). Compositionality of differential privacy allows construction of complex analyses from primitive analyses and to measure the privacy risk of the resulting analysis. A fundamental composition principle states that the overall differential privacy risk of two queries M_1, M_2 with differential privacy risk of ϵ_1 and ϵ_2, executed sequentially, is safely overestimated as $\epsilon_1 + \epsilon_2$. The decomposition of analyses, as suggested in

[1] The ratio between responses that correctly reflect the answer to responses that incorrectly reflect the answers has to be e^{ϵ}.

[2] Alternatively, the privacy cost or the privacy harm

the foundation of PreTPost [3], simplifies the correctness argument of algorithms and allows users to verify their privacy impact (Section 3.2). We study how budget enforcement, as one possible user privacy preference, is encoded in the framework as a policy. We finally investigate threat from a malicious analyst that aims to use other possible channel to ex-filtrate data (Section 4 and 5).

Scenario Note 1

As a running scenario for illustrating the use of PreTPost, we consider an Internet Service Provider (ISP) aiming to study router configurations of their users. The analyst or the curator is working for the ISP and users' data is collected from routers in private homes.
The goal is to study user demographic of users with security mindset to overall users in order to make data backed decision on default configuration and features in the future product release.
We ported and deployed the framework in embedded devices by packaging it for the OpenWRT Linux. The OpenWRT project is aiming to provide a light-weighted Linux distribution for embedded devices and routers with limited storage capacity and computation power.

2 Communication Between Curator and Users

In the centralised model the plain data is simply collected, stored and processed. However, from the user (and sometime legal) perspective the collection of data, that is not necessarily and immediately needed, is not acceptable. As a solution to this privacy concern the local model tries to minimize the data and ensure that only data that is necessary for carrying out the current analysis is extracted and transmitted to the curator. The curator may have instructed to throw away the data as soon as the data is aggregated. This means the curator and users have to communicate and agree for every analysis. In the Section 4 we study who the user can delegate the query acceptance process to policies but in this Section we only look into different types of communication between users and the curator.

The current implementation of PreTPost framework runs a HTTP web server (based on Flask [9] python library) that provides responses to the queries as XML documents to the curator initiating the connection.

The first type of communication between the curator and users involves announcing user's privacy preference in the form of policies. There are two kinds of policy, the users' *private* policies are not accessible to the curator, however the curator can either request or query users' *public* policies. Figure 1 shows a curator that learns about user policies by

requesting users' public policies. As we see (Section 4) the public policy can be arbitrary complex, determining the purpose of a policy is challenging for a curator. Therefore, a simpler option the curator can query the policy by sending the query and examine if it is acceptable by a user as in Figure 2.

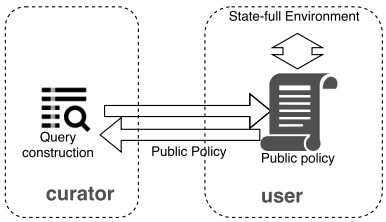

Fig. 1. Requesting the policy

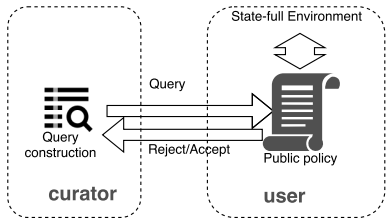

Fig. 2. Querying the policy

Policies reflect user's privacy preference and expectation. The result of a *public policy* is assumed to be non-sensitive while this is not true for a private policy. A simple policy the gives perfect privacy may reject all queries to exclude the user from all analyses. The public policy information is used to estimate the number of participants that are willing to contribute their data and their acceptable level of risk. The policy guides the curator adjusting the value for the privacy risk (ϵ) for the current and future analyses. The curator may use different strategies to decide on the level of risk for individual user. As an example use an identical risk value for all analyses or sample a subset of the population for the analysis.

Scenario Note 2

The execution of the following Python code by the curator requests users' public policy.

```
import curatorlibdp as cdp
analysis = cdp.ReqResList()
print(analysis)
```

As the result the public environment of the user device and the public policy is send back to the curator. The curator then counts the number of participants and learns about their privacy expectation.

3 Query Construction in the Curator Side

The curator first has to decide on the minimum acceptable level of accuracy (utility) needed to achieve meaningful and acceptable overall results before transmuting the queries to users. The mean squared error of the estimate or a confidence formula in shape of $Pr[X \geq \alpha] \leq \beta$ is usually used to perform accuracy analysis to express with the probability

of $1-\beta$ the random variable X is bounded by α. In the mentioned formula the X is a random variable that represents the relation between the noisy differentially private response and the correct value such as numerical difference between them ($X = |\hat{f} - f|$). For a particular β and ϵ, the accuracy usually increases as the number of users that are participating increases.

Scenario Note 3

As shown in [4] the accuracy of the procedure that collect probabilistic randomised responses ($r_i \in \{0, 1\}$), from n users all using the privacy parameter ϵ for determining the frequency (f) of underlying answers is:

$$\Pr_{\hat{f} \leftarrow \frac{\sum_{i \in n} r_i}{n}} \left[\left| \frac{1 + e^{\epsilon}}{e^{\epsilon} - 1} (\hat{f} - \frac{1}{1 + e^{\epsilon}}) - f \right| \geq \frac{1 + e^{\epsilon}}{e^{\epsilon} - 1} \sqrt{\frac{\log(2/\beta)}{2n}} \right] \leq \beta$$

Accuracy is only one side of the story; on the other hand, users may expect a privacy expectation that conflicts with the curator's expectation of accuracy. This is a trade-off that the curator has to resolve before going further with query construction.

Scenario Note 4

To achieve a high accuracy result, the curator in this study decides to perform the analysis with the high privacy risk of $\epsilon = 1$. They notice (using Chernoff bound) that if the result requires to be in the determined bound with 95% probability to be truth-worthy, with 1,000,000 users [a] the result is bounded by: $2.16r - 0.5591 \leq f \leq 2.16r - 0.5619$

However, they quickly realise that not all users are willing to participate in such a risky analysis and the query depletes the privacy budget for many users. This means the company will not be able to perform any further analysis on many users later, hence noisier results in future. Therefore, the curator relaxes their expectation of accuracy by setting a lower value of ϵ or β.

[a] Numbers for this example are taken from [4]

As explained in [3] and seen in Figure 3, each analysis needs to be decomposed into pre-processing, randomised transformation and post-processing. Each library comes with skeleton codes for constructing a query with these components. Currently the system is equipped with the libraries for the following analyses:

- Randomised response method to measure the frequency of answers to a predicate (a true or false statement)

– RAPPOR permanent method to measure frequency of strings and categories that not enumerable
– Duchi method to measure central tendency or mean value of real numbers. In the framework we used the fixed version of the algorithm introduced by Nguyên et al.
– Harmony method to measure the mean value as an improvement to the Duchi's method.
– Laplace method is a standard and well-known method to anonymise real numbers.

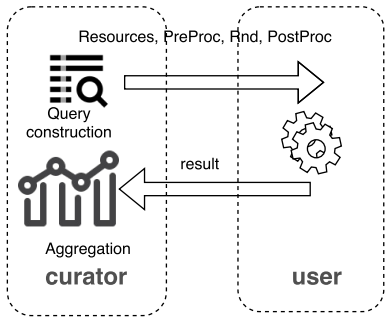

Fig. 3. Request and Response

To use the skeleton codes, the curator needs to know the type of output (e.g. string, double, category of values) and the type of analysis (e.g. mean, frequency) to choose the right skeleton library and perform as below:

– Modify the pre-processing (and sometimes post-processing) for collecting, analysing and transforming the sensitive data into the right format for randomised transformation phase.
– Determine the resources that are needed (time, computation and data resources) for pre and post processing
– Adjust the privacy risk parameters of the randomised transformation (e.g. epsilon or values in transformation matrix)

Parameters Pre and post-processing programs take a file name for storing output data as the first command-line parameter. However, within each query extra parameters can be passed implicitly (within the programs) or explicitly through extra command line arguments. One usage of the extra argument is sharing a random seed between pre and post-processing as used for *Basic Randomiser* in [3].

Resources The pre and post-processing programs may need access to data from several resources to perform the analysis. In addition to sensitive input files, the analysis may use computational power, memory, IO

and other resources that are presented as resources in the users' devices. These resources have to be known and listed in a query in advance to help users make smarter decision before accepting the query. As we explain in Section 5 PreTPost uses this information to prevent side channel attacks.

PreTPost also has an environment for data storage that is publicly accessible by the curator in which a query and its result is stored. In the next Section we discuss the access modes that the pre/post-processing have to this storage we postpone the enforcement of this access mode to Section 5.

In the local differential privacy model the data is not stored on the curator site. Therefore, the curator has to decide on an analysis and query the users for the required data. In the next Sections we study how data collection and analysis are done using pre-processing and post-processing. Please note that the pre/post-processing programs has write-only access to the are listed as outputs and inputs are read-only (Therefore no read from output resources).

3.1 Pre-processing

The pre-processing program collects, analysis and processes the sensitive data and optionally reduces its dimension. The only consideration for pre-processing is to ensure that the output is in the domain of randomised transformation. A naíve pre-processing program collects different pieces of data and transmit them. However, a better approach is to process the data as much as possible and reduce the dimensionality before transmission. Intuitively, with less data collection less data anonymisation is needed, thus less privacy risk.

To summarise the input and outputs are:

Inputs:

- Pre-processing argument
- The sensitive file resources
- Public environment

Output:

- an XML document containing the results of the analysis.

Restricting the output to an XML file forbids the curator to directly communicate to the curator or leak information through other channels such as public environment.

Scenario Note 5

Users password strength and whether users have customised the firewall configuration file are measured to determine whether a router is configured securely or not. The pre-processing listed below check if the password length is larger than 8 characters and if the firewall configuration file is modified in the last 90 days to categorise the user as either a secure-minded user or a normal user.

```
pw = readPassword("passwords/list.txt")
today = time.time()
filetime = os.path.getmtime("firewall/config.cfg")
secure = len(pw)>8 and today-filetime>90*24*60*60
libdp.toXML(outFile, [secure])
```

In Scenario Note 9 we explain how access to these files, the time-out, memory and computation power that the pre-processing needs is stated in the query.

3.2 Randomised Transformation

Since from pre-processing uses data from sensitive sources the result has to be treated as sensitive and has to be randomised before collection. Several methods for data randomisation have been introduced, such as using laplacian [2], geometric [6] or exponential mechanisms exponentialMechanism2007.

In order for the user to verify the privacy impact of an analysis, the curator has to mention the privacy cost of the analysis in a standard and a verifiable manner. Currently two methods were enough to encode current local algorithms, Laplace method and encoding using a stochastic transformation matrix. In this paper we do not dive deep into details of these techniques but instead focus on one general technique used for discrete input and output using transformation matrix. In transformation matrix, each attribute in the row represents one possible input and the values exhibit the probability that the input is mapped to the attribute of the same column. Therefore, the values in each row have to sum up to one. There is one row for each possible input, and one column for each output, where the value in row i and column j is the probability that input v_i will give the output v_j. Note each row is a probability distribution (non-negative reals which sum to 1).

$$\mathrm{T} = \begin{array}{c} \\ \vdots \\ i \\ \vdots \end{array} \begin{array}{c} \begin{array}{ccc} \cdots & j & \cdots \end{array} \\ \begin{pmatrix} \ddots & \ddots & \ddots \\ \ddots & T_{i,j} & \ddots \\ \ddots & \ddots & \ddots \end{pmatrix} \end{array}$$

We interpret a stochastic matrix T as a probabilistic function, writing $T(v)$ for the action of T on value v, where $\Pr[T(v_i) = v_j] = T_{ij}$. We also assume that the input and output domains are the same.

The matrix corresponding to randomised response is a subclass of staircase matrices [5].

Scenario Note 6

To build a staircase matrix with the privacy cost of ln(2) the `stairCaseMatrix` method from the libdp library can be used.

```
epsilon = 2
t= libdp.stairCaseMatrix(epsilon, ["-1", "1"])
print(t)

attribute,-1,1
-1,7.38905609893065,1
1,1,7.38905609893065
```

Once the matrix is constructed in the curator sidem the privacy risk of randomizing values with a transformation matrix is verified in the user side as $ln\left(\max_j\left(\frac{\max_i T_{i,j}}{\min_{i'} T_{i',j}}\right)\right)$ that is derived from the definition of differential privacy[3].

Scenario Note 7

The `measureEpsilon` function and `loadTransitionMatrix()` function are used to load a matrix from a file and measure its differential privacy cost (risk):

```
m, f = dp.loadTransitionMatrix('matrix.txt')

print(m)
{('1', '-1'): 0.11920292202211755,
('-1', '-1'): 0.8807970779778824,
('-1', '1'): 0.11920292202211755,
('1', '1'): 0.8807970779778824}

print (dp.measureEpsilon(m, f))
2.0
```

3.3 Post-processing

The output from the randomised transformation is considered public and can be transmitted to the curator for data processing. However, for performance reasons the curator may delegate some processing to the users.

[3] where $x/0 = \infty$

Inputs:

- Post-processing argument
- Randomised results
- Requested resources except the sensitive resource
- Public data

Output:

- The final deliverable output.

Note that the post-processing has access to all public resources except the sensitive data.

Scenario Note 8

The listing below shows a post-processing for randomised response that scale the result based on the epsilon that is given as a parameter.

```
data = libdp.fromXml(inFile)
epsilon = float(sys.argv[3])
scale = (math.exp(epsilon)+1)/(math.exp(epsilon)-1)
out[0] = float(data[0]) * scale
libdp.toXml(outFile, out)
```

3.4 Query Transmission and Data Collection

Once the different components of a query (pre, T, post) are constructed, the curator sends the query to users.

Scenario Note 9

The code snippet below exhibits the communication between users and the curator. The curator running the code sends identical queries to all users.

```
analysis.runIdenticalQuery(
  pre='curator/analysis/randomized-response/preproc
      .py', preParam=epsilon, resources[timeout:2,
      cpu:1, memory=100MB, mount:configData|
      passwordData] ,
  typeDataList=[('analysesMatrix', t)],
  post='curator/analysis/randomized-response/
      postproc.py', postParam=epsilon, resources[
      timeout:2, cpu:1, memory=100MB] ,
)
```

The curator knowing the time-outs, collects the data from users when the policies, randomised transformation, pre and post-processing are fully executed.

Scenario Note 10

```
% wait for 'time-out' seconds
print(analysis.ReqReslist)

[reqId:1505509340404 uri:http://user1:5000/ budget
    :1.2 results:{0=-1.3130352854993312,},
reqId:1505509340419 uri:http://user2:5000/ budget
    :0.7 results:{0=1.3130352854993312,}]
```

Finally, the curator aggregates the results that are collected from users.

4 Query Execution on the User Side

As we briefly explained, the curator dynamically designs analyses and transmits them to the users for execution and in the user side a HTTP server is running to respond to curator's query. To give the control back to the users, users decide on their public and private policies.

With the assumption of an untrusted curator and the promiscuous and possibly malicious nature of pre-processing and post-processing, the curator may exploit the user system or use side channels to leak sensitive information to the curator. The PreTPost framework provides isolation and resource constraints of the received analyses. Each component that is executed in the agent machine needs its specific isolation considerations. In this Section we take a look at policies on the user side and the sandboxing considerations of each component.

4.1 The Public Policy

As the name suggests the public policy announces the user's policy preference such as type of analyses that the user is interested to participate in. Users encode their choice of accessible resources and the level they are eager to participate in analyses in a public policy function that rejects or accepts queries. The curator benefits from the knowledge of policy by having an estimate on number of participants and the accuracy to design the current and the future queries more wisely. Making sense out of an arbitrarily complex python script as a policy is difficult. Therefore, it may even be accessible to the curator to be queried. In this case once a query is provided, troubleshooting why a query is rejected and how to design a better query is also troublesome.

Scenario Note 11

Suppose a malicious analyst in the ISP tries to find an individual's password by querying every bit in the bit representation of the password. Since the password is a long string of bits and the result is probabilistic he decides to query each single bit multiple times knowing that the correct value for each bit will appears roughly 3 times more often than the incorrect value.

Budget is the value that specifies the threshold on risk that a user accepts. The basic budget enforcement policy uses the sequential composition to ensure that the sum of risks for queries (ϵ_i) that are accepted does not exceed the budget. ($\epsilon_1 + \ldots + \epsilon_1 \leq \mathcal{E}$).

Scenario Note 12

A user, who is willing to participate in analyses that asks for firewall configuration adds the resource to allowed resources list (`allowedRes`). To accept a request, the policy checks if the set of requested resource (`reqRes`) is a subset of the allowed resources. The basic budget enforcement policy uses privacy cost accounting and access control over resources to decide whether to reject or pass queries. `measureRisk` uses `measureEpsilon` (Scenario Note 7) and other functions to identify the total risk caused by a query.

```
def policy(reqId):
    allowedRes = ['configData', 'passwordData']
    budget = 0.4
    accCost = ...
    cost = libdp.measureRisk(reqId)
    reqRes = libdp.requestedResources(reqId)
    return (cost+accCost < budget) & (reqRes <=
        allowedRes)
```

Since the public policy is chosen by the user, we assume that it is trusted and not harmful. However, users may mistakenly make data dependent decisions in their policy and a curator can take advantage of this. Background information connects and correlates different pieces of information in ways we cannot predict in advance. Therefore, under differential privacy all data points have to be treated equally sensitive. As an example one may decide on the acceptable level of privacy (ϵ) based on data itself.

As a more concrete example a user may write a policy that reject queries on password file when their password is too short. The rejection of a query by itself signals the curator about the weak password.

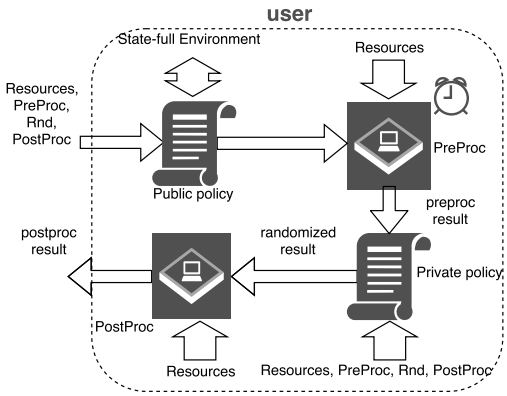

Fig. 4. Query Execution

If a query is accepted in the public policy, it is passed to pre-processing phase for execution. In the next section we will see how the private policy decides about the result of the pre-processing phase.

4.2 Private Policy

In some specific cases the choice of policy may be revealing and therefore needs to be protected. The purpose of the private policy is to encode the requirements that are sensitive to be expressed in the public policy in a protected way. As an example the private policy may have access to the private data and make data dependent decisions. The private policy comes after the pre-processing but before the randomisation and decides on acceptance or rejection of a query without invalidating the differential privacy guarantee. Additionally, it strengthens the plausible dependability by adding another level of uncertainty to the answer. As an example it may replace all private values with dummy values if the user, for privacy reasons, decides not to respond to the queries. This gives users even a stronger plausible deniability.

Scenario Note 13

```
def policy ( reqId ) :
    return True
```

4.3 Randomised Transformation and Domain Enforcement

While procedures up to this point may be probabilistic, they are not directly used for enforcing differential privacy. In this phase the randomisation specified in the query is applied to the data to anonymise the result. Note that the curator can only specifies the type and parameters of randomised transformation.

An erroneous or possibly a malicious pre-processing may generate an output that is not in the domain of the randomised transformation or causes observable exception or behaviour. The domain enforcement function ensures that the randomised transformation receives inputs in the valid domain and format and generate outputs in the valid range and format. After this phase the probabilistic output is ready to be passed to the pre-processing.

Scenario Note 14

Now that the analyst realised that extracting the password silently via statistical analysis is not possible, they plan for other direct attacks.

- Use the rich functionality provided in the supported language to send the password as an email as soon as pre-processing or post-processing are executed.
- Use low privacy cost queries but communicate using side channels (such as execution time or exception handling of the pre-processing).

5 Isolation

Queries that come from the curator may have malicious intent. Therefore, the sandboxing has to ensure that information leakage, explicitly by sending it directly to the curator or implicitly via side channels, is not possible. In this Section we study the purpose and level of isolation of each component in the user side.

Public Policy The public policy is decided by the users and assume trusted. The framework makes it impossible to mistakenly use sensitive data.

Pre-processing First and foremost, the pre-processing has to be isolated to avoid direct transmission of data from protected environment out to the curator. Another observable result of the pre-processing, apart from the standard and expected output, is the execution time. To avoid the timing side channel, the sandboxing enforces constant execution time for all analyses. Two approaches are imaginable for performing analysis working on data over a period of time. Collecting data and then performing the analysis, or running an analysis in background over the period of time. In both cases the execution time has to be known and specified in the request in advance.

The allocation of resources such as IO, memory and CPU that may be observable from outside of the system, or in the case of parallel query execution outside the sandbox, has to be reserved before the execution. As an example a malicious analysis can make a system unresponsive dur-

ing its execution by requesting too many resources leaking information out the system. In this setting we assume that CPU usage and memory allocation patterns are not visible to the curator. Therefore, the resource request has to be negotiated and agreed with the public policy, and reserved in advanced before execution.

Private Policy and Domain Enforcement The sandboxing is in place to ensure the constant execution time of the private policy, the domain enforcement procedure and randomised mapping. This phase is considered trusted to read sensitive data but sandboxed to block information leakage to the outside world.

Post-processing The input to this phase is considered differentially private already and therefore this phase can be done in the curator side. However, since the post-processing comes from the curator, the sandboxing has to ensure that the code does not get access to the sensitive data.

6 Experiment

As explained in Section 4, a process can be isolated using the sandboxing features provided by Linux. We are aware of possible side channel in the implementation of nsjail or other component of the prototype implementation of PreTPost, and the sandboxing solution chosen for PreTPost is the best effort.

Nsjail [7] that is used to control access over CPU, memory and I/O resources takes advantage of isolation and security capabilities of Linux kernel such as name space subsystem, resource limits, and the seccomp system call filtering[4].

Communication between processes when the system deals with parallel execution of analyses and the timing property when they are executed concurrently may leak data. Therefore, the parallel execution of analyses is forbidden in the current implementation of PreTPost.

As an alternative solution for cases where the power, CPU, cache or memory usage pattern may reveal sensitive information, one may consider using a separate computing unit for the computation performed by the PreTPost framework. As experimented a cheap $5 Raspberry-pi zero is powerful enough to physically isolate all the computations from the main processing power.

As explained in Section 3 the timing and resource properties of an analysis is specified in the curator side and transmitted with the query to the user. Enforcing the time-out limit is necessary to prevents side channels but having a tight bound requires the aggregator to be aware of processing work load on the user side. Although overestimating the

[4] We can alternatively, use a light sandboxing tool named ujail for architectures (MIPS) that are not fully supported by nsjail.

execution time is necessary for systems with shared processing resources or when the computation power in the agent side is not known, it negatively affect the real time behaviour of the analysis. Even though little effort is placed on optimizing the PreTPost framework and reducing the resource usage, it can still be executed in a system with relatively low computation power and memory.

7 Future Work

To comply with some regulations (such as GDPR) the PreTPost can be extended with more features.

Purpose of analysis Knowing the type of data that are collected from the user is usually not enough to determine the purpose of an analysis. Whether the data is used in academic study or used for marketing purposes may be explicitly specified with every query.

The policy on the user side checks if the purpose is among acceptable purposes specified by the user for deciding on participation in an analysis.

Consent A curator can collect consent by asking the framework to sign the queries that it accepts or having the user to sign the policy.

Right to be Forgotten The curator has to disassociate the user identifiers (e.g. IP addresses) with their responds after data collection, otherwise if data storage with user identifiers are needed, the users have to be able to remove their data at any point.

8 Conclusion

In this paper we have presented the practical aspects of PreTPost differential privacy frameworks. Using a concrete example of an Internet Service Provider, construction of a query in the curator side and its verification by policies on the user side (Home routers) are studied. We finally looked into the execution of the query and the isolation consideration that are required to avoid data leakage in different component of the system.

References

[1] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. Calibrating noise to sensitivity in private data analysis. In *Proceedings of the Third Conference on Theory of Cryptography*, TCC'06, Berlin, Heidelberg, 2006. ISBN 3-540-32731-2, 978-3-540-32731-8.

[2] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. Calibrating noise to sensitivity in private data analysis. In *Proceedings of the Third Conference on Theory of Cryptography*, TCC'06, 2006.

[3] Hamid Ebadi and David Sands. Pretpost: A transparent, user verifiable, local differential privacy framework. 2018.

[4] Marco Gaboardi. Topics in differential privacy, lecture notes. `http://www.acsu.buffalo.edu/~gaboardi/teaching/CSE660-fall17.html`, 2017.

[5] Q. Geng, P. Kairouz, S. Oh, and P. Viswanath. The staircase mechanism in differential privacy. *IEEE Journal of Selected Topics in Signal Processing*, 9(7):1176–1184, Oct 2015. ISSN 1932-4553.

[6] Arpita Ghosh, Tim Roughgarden, and Mukund Sundararajan. Universally utility-maximizing privacy mechanisms. *CoRR*, abs/0811.2841, 2008.

[7] Google. Nsjail, a light-weight process isolation tool. `https://github.com/google/nsjail`, 2017.

[8] Thông T. Nguyên, Xiaokui Xiao, Yin Yang, Siu Cheung Hui, Hyejin Shin, and Junbum Shin. Collecting and analyzing data from smart device users with local differential privacy. *CoRR*, abs/1606.05053, 2016.

[9] Armin Ronacher. Flask. `http://flask.pocoo.org/`, 2016.