Thesis for the Degree of Doctor of Engineering

Dynamic Enforcement of Differential Privacy

Hamid Ebadi

Department of Computer Science and Engineering
Chalmers University of Technology
Göteborg, Sweden 2018

Dynamic Enforcement of Differential Privacy
Hamid Ebadi
ISBN 978-91-7597-685-3

Doktorsavhandlingar vid Chalmers tekniska högskola
Ny serie nr 4366
ISSN ISSN 0346-718X

Technical Report 153D
Department of Computer Science and Engineering
Software Technology

Department of Computer Science and Engineering
Chalmers University of Technology SE-412 96 Göteborg
Sweden
Telephone +46 (0)31-772 1000

Printed at Chalmers
Göteborg, Sweden 2018

Abstract

With recent privacy failures in the release of personal data, differential privacy received considerable attention in the research community. This mathematical concept, despite its young age (Dwork et al., 2006), has grabbed the attention of many researchers for its robustness against identification of individuals even in presence of background information. Besides that, its flexible definition makes it compatible with different data sources, data mining algorithms and data release models. Its compositionality properties facilitate design of "differential privacy aware" programming languages and frameworks that empower non-experts to construct complex data mining analyses with proven differential privacy guarantees. The goal of this research is to introduce new (and improve the current) differential privacy backed frameworks, prominent both in utility and flexibility of use. We study dynamic enforcement of differential privacy both in the *centralised model* in which a trusted curator process data stored in a centralised database and the *local model* with no trust on the third party.

For the centralised model the thesis mostly focuses on the privacy impact of the basic building blocks used in these frameworks, proving correctness of the system built upon them. With respect to accuracy, we present *"personalised differential privacy"* as an improved method of enforcing privacy that provides better data utilisation and other benefits. In this setting, individuals take control of their privacy requirements rather than being seen as a part of a database. As a result, they can opt-in to a database with their expected privacy level and optionally opt-out later. We further study the privacy implication of other building blocks such as different kinds of sampling and partitioning.

For the local model we propose a general framework in which the users can verify the recieved analyses and with a flexible policy express their privacy preference in different forms such as enforcing their personalised privacy budget.

Acknowledgements

Thanks to my families for their unconditional love. Thanks to my supervisor, David Sands, for his patience, true kindness and his support. Thanks to Gerardo Schneider and Thibaud Antignac as my co-supervisors and Andrei Sabelfeld as my examiner for their help and guidance throughout my study. And finally to my friends in Sweden for the great times that we have spent together and all my friends at Chalmers for all the inspiring discussions during fika times.

Funding

This work was partially supported by a grant from the Swedish Foundation for Strategic Research (SSF).

Table of Contents

Introduction

Personal information is collected starting from morning; a transportation record is stored in a database when you use your contact-less bus card. Data is collected as the bus passes through mobile phone cells, when you are busy checking your emails or when you interact with others via social networks. Before you declare the time and your attendance in the office using biometric methods, you might use your credit card to buy a coffee from a shop protected by a CCTV camera. Different kinds of information are collected and stored in different databases, possibly in different geographical locations. Some of this data is legitimately collected to be used for strict and predefined purposes. In the above scenario, your salary or your transportation cost is calculated from the collected data.

Besides the desirable and legitimate processing of data, each service provider involved in this scenario can benefit from collected information in different ways. It is common for companies to use this data to learn about their customers in order to predict their behaviours and improve the service. As an example, the transportation company can change the number of buses in different areas, adjust the trip frequency according to the time of day, or place more stations within areas with higher traveller density.

Using personal data for such purposes seems not only benign but also desirable; however, setting a proper border on how data should be handled is a question that needs more scrutiny. Knowing that an analysis is constructed from individuals' personal information confronts us with the crucial question: is it possible for someone to learn about an individual by looking at the results of statistical analyses. Back to our example, looking at the schedule and frequency of buses, can travel information of an individual be inferred?

Even though the leakage in this specific case seems to be insignificant, how can this leakage be quantified? What if one bus stop is only used by one individual? While using the valuable information stored in a database is tempting, this information has the most value for those who own it. Any mistake or mishandling of these sensitive private data can easily cause chaos.

Investigating how much information can be extracted from the result of a statistical analysis helps us to identify possible threats, and to provide better assurance for users' privacy.

1 Why Does Privacy Matter?

Our personal information is valuable, but privacy, like security, becomes an important issue when a breach occurs. Considering websites and businesses that provide free services, solely by trading users' personal information, we see the immediate value that our personal information has. Unfortunately, with the current state of Internet and society, where people and Internet do not forget, the consequences of privacy violation in the long term are not rectifiable.

No one can quantify the physical or emotional harm resulting from a privacy breach since the type of possible harm, its duration and its effect on different people is not completely known. Data revealed as a result of a privacy breach may persist as long as the lifetime of its owner or even longer in the case of Genomic data. This explains why possible harm caused by privacy breaches should not be underestimated. While eating habits and the amount of alcohol an individual consumes, may be interesting for an advertising company, it is important to remember that the advertisement companies are not the only people who can take advantage of this information. This information can likewise be used in the process of recruitment for your next job, to effect the outcome of university selection, or be used by health insurance companies.

Similarly, while the leakage of a database that keeps individual coffee orders may at first sound insignificant, it reveals the person's location, or in other words, his absence from his office during working hours. The number of coffees that the person bought can reveal whether he was alone or accompanied by another person. Presence of two persons in the coffee shop at the same time may disclose an affair, special communication or friendship between them. Knowing the political orientation of one side may reveal more about the purpose of the meeting. Progress in science may reveal a pattern between coffee drinking and a mental/physical problem which gives new meanings and dimensions to this plain information record. As it can be seen, any kind of information that individually looks harmless can be abused if it is linked (possibly in the future) with meaningful background information. The harm of a privacy violation may go beyond the individual that is the subject to the privacy breach or even beyond their lifetime. In 1951 after Henrietta Lacks died from aggressive form of cervical cancer, her cells (also known as HeLa cells) were cultivated in a laboratory. Progress in genome sequencing suddenly introduced a new privacy threat to her family as HeLa cells' genome was published on Internet [40]. The genomes have much similarity with genes that she passed on to her children and grandchildren living today. Hid-

den in the sequence is potential biomedical information about Henrietta's descendants, such as their risk for getting Alzheimer's disease or other kinds of disorders. Recent studies [29, 28] have shown that social media contents can be used to infer other information that are not voluntarily expressed such as age, sex, political view, intelligence, sexual orientation and preference.

Privacy is a broad topic; in this thesis we focus on privacy in statistical databases. Since privacy is better understood when the violations are investigated, in the next section we review some failures caused by careless releases of databases.

Failures of Database Anonymisation If there is no privacy concern, public interest encourages information disclosure as a way to learn more about behaviours and common patterns. To achieve privacy, anonymisation (de-identification) – that is the process of removing personally identifiable information (PII) from raw data – is often used. A *quasi-identifier* is the unique key (such as social security number[1]) used to identify a person in a database. While the idea of removing quasi-identifiers from database for anonymisation seems to be promising, recent efforts in data anonymisation have ended up in scandals. In what follows list some of the most well-known privacy failures of anonymisation efforts is listed.

AOL Search Query Data Scandal When AOL released a huge dataset of search queries belonging to 650,000 users, the company claimed that they had applied anonymisation thoroughly as the IP addresses were removed and user IDs were replaced with random numbers. The random numbers avoid identification of individuals while at the same time allowing researchers to correlate different search queries that belong to individuals. While preliminary thought suggests that these data are random terms issued by random people, looking at search queries one can notice a person's feelings and thoughts in different social or emotional conditions. Using these, it is possible to identify a person who wants to commit suicide, someone who looks for a restaurant nearby and someone looking for medication by searching symptoms.

These terms can easily reveal someone's identity when a user searches for a friend's name, a nearby location or a rare disease. In these cases, the searched terms can be linked together to single out and expose the faces behind them [4]. Once the person is identified, her/his other queries can reveal his/her deepest feelings, passions or secrets. To show the privacy risk, the New York Times revealed the face behind the user No. 4417749 who searched for "numb fingers", "60 single men", "homes sold in shadow lake subdivision gwinnett county georgia" , "landscapers in Lilburn, Ga," and "dog that urinates on everything.".

[1] Similar to the personal number in Sweden

Massachusetts Group Insurance Commission In another privacy incident, the "Massachusetts Group Insurance Commission" released hospital visit information for state employees after removing social security numbers and addresses. William Weld, the Governor of Massachusetts claimed that the anonymisation of the information protects the privacy of patients. Sweeney [42] purchased the voter registration list for Cambridge Massachusetts city and linked these two databases to find the governor's health record. She narrowed down the possible records that may belong to William Weld to three, using only birthday and gender. Finally, using the five digits ZIP code, Weld's medical record was uniquely identified revealing information about his ethnicity, visit dates, diagnosis, procedures, charges and medications. While name and social security number are clearly considered the key to identify a person, Sweeney's research concluded that combination of ZIP code, birthday and sex can alternatively be used to uniquely identify 87 percent of the population in the US.

Netflix Competition The process of correlating and linking data to individuals is surprisingly effective knowing the person's preferences or interests. User preference, like ratings given to a movie, that has never seriously been considered personally identifiable information, is shown to be informative enough for re-identification purposes. When Netflix announced a prize for an alternative algorithm that can predict user ratings for movies by mining current users' ratings, a real dataset of users' ratings was provided to let the contesters train, test and measure the quality of their algorithms.

The dataset had 100,480,507 ratings that 480,189 users gave to 17,770 movies. To anonymise the dataset, user IDs were replaced, some ratings were randomly changed, added or deleted and some rating dates were also modified. Even with this level of anonymisation, Narayanan and Shmatikov [33] demonstrated that movie rental details (time and the given rating) of three movies are enough to successfully link an individual to his records in the database. They compared the dataset against publicly available IMDB ratings as the source of background information. As a result of their research, one can conclude that the combination of spatial and high dimensional attributes can make re-identification easy.

As the result of these privacy scandals, AOL took down the user search database several hours later. Under the pressures of a lawsuit and Federal Trade Commission questioning, Netflix also decided to shut down the contest. These countermeasures, as usual, were not effective as both databases have been mirrored immediately and distributed over the Internet.

Background knowledge and cross-correlation with public databases is a common way to re-identify individuals when obvious quasi-identifiers are removed. Re-identification is done when the result of linking our data

with a background database leaves out few possible individuals for each anonymised record. Sweeney [42] proposed the *k-anonymity* property for the data. In k-anonymity the data provider (by manipulating data) ensures that each individual's information is indistinguishable from at least $k-1$ other individuals present in the database. While it is not difficult to see why k-anonymity totally fails in high dimensional databases, knowing possible attributes, even though not precisely, can still be informative. As an example, a dataset that has 3-anonymity with respect to the attribute disease may reveal that one person has HIV, syphilis or chlamydia. K-anonymity also fails completely when the attacker is in possession of additional knowledge. This background information can be a database that is released with a k-anonymity guarantee which shows k-anonymity does not compose with itself.

As explained earlier in this section, in non-interactive data disclosure, sanitized data is released after applying some anonymisations (also known as de-identification) once and for all. Removing obvious identifiers, sub-sampling and perturbing values are used to construct the "synthetic data". Unfortunately, the data released in non-interactive methods is only suitable for the particular class of data analyses that it is released for. In the next section, the possibility of a more fine-grained data release method, flexible enough to be used in different scenarios that supports interactive systems, will be discussed.

2 Differential Privacy

The consequences of failures in anonymisation show the challenge of statistical data disclosure. It seems the perfect privacy scenario from every individual's perspective is achieved when the individual opts-out from the analysis and removes their record from the database. Privacy for everyone means that all records in the database have to be removed which results in an empty database with no analysis. This is in line with another expectation of privacy that the access to a database should not reveal anything about an individual that cannot be gained without the access. Dwork [8] shows the absolute disclosure prevention is impossible in presence of auxiliary information that may be totally irrelevant to the database. The impossibility result can be demonstrated with a simple scenario. Assume the money that Zlatan Ibrahimović spent for alcohol in last midsommarfest[2] is considered to be secret. If auxiliary information reveals that the money could feed 100 people in Fiji for a week, access to the database of expenses in Fiji reveals the sum of the purchases, regardless of Zlatan's presence in that database. The fact that we cannot protect his privacy is inevitable and leads to a new formulation of privacy known as *differential privacy* that makes no assumption about the auxiliary information. This means even a computationally powerful ad-

[2] midsommarfest is a Swedish celebration held to welcome summer

versary that has information about all records in the database except the target individual is not able to learn more from the individual beyond the specified limit.

Other methods of anonymisation usually suffer from two problems. The first problem arises in corner cases in which a few individuals are effectively singled-out in analyses. As the number of individuals participating in an analysis decreases the effect of each individual and subsequently the information leak about those individuals increases. Each independent analysis that is done independently leaks information too. One major difficulty is quantifying the information leakage for each analysis and the total information leakage caused by composition of several analyses.

Having all this in mind, Dwork et al. [10] formulated differential privacy as indistinguishability of outcomes when an analysis is performed on two similar databases. The *hamming distance* is commonly used as a similarity measure for databases and closeness of probability of outcomes measures the indistinguishability. A computation Q over these datasets provides ϵ-differential privacy if it keeps the ratio of probabilities (A and B as shown in Fig 1) of observing any outcome lower than e^{ϵ} when it runs on two neighbouring databases D and D' (with hamming distance 1). More precisely the definition is:

Definition 1. *A randomized function Q gives ϵ-differential privacy if for all $S \subseteq Range(Q)$ and all D and D' where they are neighbours, we have:* $|\frac{Pr[Q(D)\in S]}{Pr[Q(D')\in S]}| \leq e^{\epsilon}$

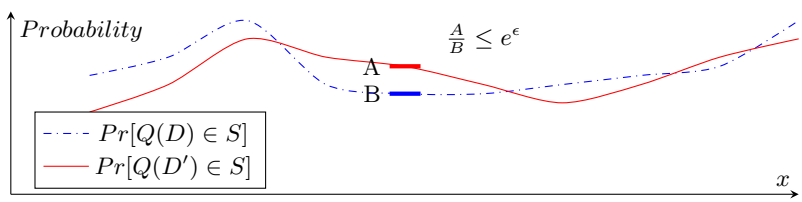

Fig. 1. Outcome Probability

Properties As it can be realised from the definition, differential privacy is flexible and not restricted to a database schema or the type of output. The basic principles used to construct analyses over the Netflix tabular database can be used to query more sophisticated data structures like graphs used to represent interactions in social networks. Differential privacy not only protects the value of data points but also protects the existence of individuals in a database. Imprecise responses give differential privacy the desirable deniability feature even with possession of everyone else's personal information. Furthermore, differential privacy has a

strong composition property that makes analysis construction from basic components possible. Something that is crucial for building tools and frameworks.

The epsilon parameter (ϵ) is a value to quantify the privacy loss or harm. The privacy parameter has no unit but enables us to compare the effect and risk of two analyses on individuals' privacy. Privacy is a fuzzy concept and different people put different values to their personal data. To understand the privacy risk better, consider the following scenario inspired by [35]. Assume results from a medical survey affects an insurance company's decision on the health insurance coverage. A change in the insurance coverage from participation of other people in the study is unavoidable; however, we expect that the cost (as a factor of risk) that Alice pays for participation in the study does not exceed a certain limit. Differential privacy can help us set these limits, pay compensation and reward people for participation in surveys.

Mechanism Design Randomized response is a well-known method proposed by Warner [48], later modified by Greenberg et al. [22], to give data subjects privacy in the form of deniability. As pointed out by Dwork [8], this mechanism provides differential privacy for answers given by participants in a survey. Suppose, for simplicity, that a yes-no question involves an embarrassing answer. In this method, to respect people's privacy, the interviewer instructs people to flip two coins secretly before responding to yes/no questions. Each question should be answered "yes" if both coins come up head, answered "no" if both coins come up tail. Otherwise the question should be answered truthfully. Keeping the result of the coin toss secret, only each person knows whether the answer reflects the reality or the random coin flip. The respondent can deny this answer by claiming that the coin flip was the reason for the response.

The analyst, knowing that each individual did this procedure before responding to each question can conclude that roughly half of the questions are truthfully answered. From the other half, roughly half of them ($1/4$ of the total responses) are "Yes" and $1/4$ are "No" as a result of the coin flip. The ratio of answers determines a rough estimate of the overall answers. One can further show that this procedure provides $\ln(3)$ differential privacy.

Several more randomisation mechanisms are suggested such as laplacian mechanism [10], exponential mechanism [30], geometric mechanism [21] and stair case mechanism [20]. The laplacian noise provides real values, for integer responses geometric mechanism introduced by Ghosh et al. [21] and for queries that ask for a non-numerical value, like "what is most common disease among database participants?", the exponential mechanism, developed by McSherry and Talwar [30] are applicable.

To use Laplace noise to make a noisy version of function f over a dataset x, one only needs to know the sensitivity of the function, the

maximum change in the result of the function when an item is added to the dataset.

The noisy version of function $f(x)$ that provides ϵ-differential privacy is constructed by adding laplacian noise to the result of the function as below:

$$f'(x) = f(x) + \text{Laplace}(\text{Sensitivity}(f)/\epsilon)$$

If f' is a randomised function constructed from the function f with sensitivity of 1, Fig. 2 shows the probability density function in two neighbouring datasets D and D' for different values of x.

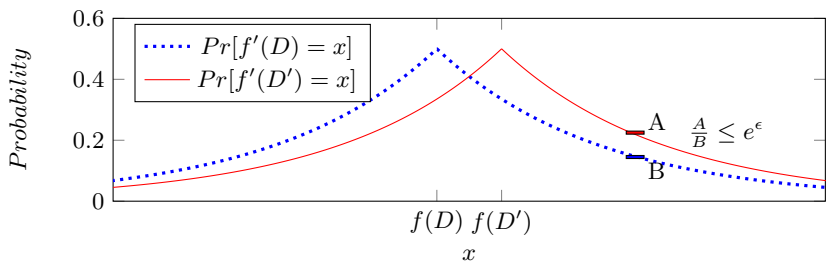

Fig. 2. Probability Density Function

Composition Principle (Sequential) Differential privacy has useful composition properties that make it possible to construct more complex analyses using basic ones. If a family of queries Q_i each give ϵ_i differential privacy, executing a sequence of the queries on the database gives $\sum_i \epsilon_i$-differential privacy. This property, named *sequential composition* is the key for building tools for doing differentially private analyses using basic primitives. In the next section we introduce some of the well-known tools in this area.

3 Variants of Differential Privacy

Apart from the standard and classic definition of differential privacy (Definition 1), different variants of differential privacy for different settings and purposes are proposed. In this section some of these variants are explained.

Centralised and Local Model Trust plays an important role in the perception of privacy. In the centralised model the users have to transmit their data to the curator. Users have to trust the curator for the safe storage of the data and correct usage of differential privacy. The centralised model is not desirable for many users and therefore the local model is seen as an important alternative. While the local model seems to be an

obvious alternative, as shown by many studies, for the same level of privacy these algorithms do not perform as accurately as the centralised model.

Bounded and Unbounded Two variants of differential privacy are commonly used in the field. Bounded differential privacy [27] deals with the case in which neighbouring databases are constructed by changing the value of one individual record in contrast with the unbounded definition with two databases, one with an extra record. This restricts the neighbouring databases to databases with the same size whereas the unbounded differential privacy can be defined for databases with a variety of sizes.

Weakening In (ϵ, δ)-differential privacy [9] (also known as approximate differential privacy), ϵ, as before, represents the effect one individual can have on the information release and δ bounds the probability of a complete privacy breach [41]. As a result, we make room for utility by allowing privacy failure in low-probable events. The case $\delta = 0$ (also known as pure differential privacy) is equivalent of classic differential privacy as explained before.

Generalising In 1970, Alan Westin conducted over 30 surveys about the privacy concerns in different areas. His report demonstrated that people are categorized into three groups: *Privacy Fundamentalists*, *Privacy Pragmatists* and *Privacy Unconcerned* that motivates the need for non-uniform privacy requirements for different individuals and items in a user profile.

Recently, in three independent works [17, 26, 1], some efforts have been made to make differential privacy user centric. We discuss our paper comprehensively in section 6.2 and here we only discuss the two subsequent papers.

Having in mind that people have different privacy preferences, Jorgensen et al. [26] proposed a personalised privacy method that can be specified at the user level. They show that the proposed personalised differential privacy framework has composition properties and a procedure is provided to convert mechanisms from classic differential privacy to their method. To summarize, the data is first sampled non-uniformly and then the right differentially private mechanism is applied. Individual tuples can have different probabilities of being sampled depending on their privacy preferences.

Even for different parts of an individual data record, there may be different privacy preferences and expectations. This non-uniformity of privacy preference brought the idea of heterogeneous differential privacy [1]. In their generalisation of differential privacy, users describe their privacy expectations with a privacy vector. The privacy vector value for each item ranges from zero for the absolute privacy to one for the stan-

dard classic privacy. Later they describe the *Stretching Mechanism* that changes the sensitivity of functions after applying the privacy vector.

Andrés et al. [2] introduce the notion of a *geo-indistinguishability guarantee*. Rather than completely hiding the user's location, this notion of privacy tries to hide locations with some level of approximation within a radius r. While hamming distance is commonly used as a metric of difference between two databases, to adopt distance to the location privacy setting Euclidean distance between locations is used. In addition, a mechanism based on laplacian noise, that let users obtain needed service even after randomised mechanism, is provided.

Different Context Assumptions : Some other research focuses on the usage of privacy in different setting and special contexts. *Pan Privacy* [12] aims to retain the privacy of individuals in a data stream while data is processed, with the assumption of an adversary that can observe the internal state of the system at any unpredictable time. Dwork et al. [11] consider a system in which the result of the same query changes because of a change in the underlying database. This is a common situation for monitoring systems in which the system continuously produces output (this may not be the case for Pan-Private systems). In this setting neighbouring (adjacent) inputs should be defined differently. In *event level privacy*, two databases are neighbours when they differ in one of the events that belongs to a user, whereas in *user level privacy* the databases may differ in multiple events all belonging to one single user.

4 Scientific Methods and Goals

In the first few sections we looked into importance of privacy, naive deanonymisation techniques and scandals caused by them. In further sections we continued by introducing differential privacy as prominent notion and studied its variants. We introduced few privacy enhancing tools and regulatory requirements aiming to improve the current state of user privacy. Among these tools we are particularly interested in programming language approaches that facilitate analysis construction as a program and enforcing the differential privacy dynamically at runtime. We improve these systems by 1) proposing a new privacy accounting schema that permits users to announce their expectation of privacy 2) add new functionalities and operations that are not necessary deterministic. 3) Shifting the trust from the analyst, who performs the analyses, to data owners.

To study and verify the correctness of these systems, we build minimalistic but formal models. These models allow us to abstract away from details of implementation and to focus on fundamental components, their behaviours individually and lastly their interactions with the rest of the system.

5 Differential Privacy, Tools and Methods

Data holders, eager to release information, are not usually privacy experts. Going through different formulas and computing the privacy cost of an analysis is a quantified real-world problem. In addition, an analysis can be arbitrarily complex. A data mining algorithm usually consists of sub-analyses, few iterations and branches on an intermediate result. This brings up the idea of using programming language techniques to ease the construction of these analyses. As a result, many differential privacy frameworks, methods and tools with variety of purposes are introduced. In this section we look at some of them.

Privacy Integrated Queries PINQ [31] is a generic framework to construct differentially private analyses for analysts that are unfamiliar with differential privacy. All programs which are written in this framework and use its APIs, automatically satisfy differential privacy. In PINQ, every data source is wrapped in a PINQ (.NET) object. From this point, the data can be only manipulated via a restricted API that only allows aggregated results to be emitted to the analyst. These primitives are divided into two major categories, transformations and aggregation.[3] Input data is usually transformed before being used in an aggregation function. Aggregation functions work on some specific types of input and these transformations shape the input accordingly. Only stable transformations are allowed in this framework. The stability of a transformation T is c if for any datasets A and A' :

$$|T(A) \oplus T(A')| \leq c \times |A \oplus A'|$$

Some transformations are intrinsically stable like mapping a function over records in a dataset, filtering records based on a predicate, and grouping records into limited groups that share a property. In unstable transformations small differences between neighbouring datasets (when a dataset contains an individual's records and when it does not) may lead to unbounded difference in the resulting datasets. To deal with unstable behaviour of operators like *join*, PINQ introduces a restricted variant with *similar* behaviour.

When data is selected and formed for a statistical analysis, an aggregation operation (like Count, Sum, Average, Median, Min, Max) is executed on it. The result of these aggregation operations are allowed to be emitted to the analyst. Aggregations in PINQ use the laplacian mechanism to add the needed amount of noise. Composition of a c-stable transformation with an ϵ-differential privacy query results in a computation with higher sensitivity that gives $(c \times \epsilon)$-differential privacy. *Function sensitivity* is a numerical value that measures the maximum order of mag-

[3] PINQ has a modular structure and can be extended with new primitives (transformations or aggregations).

nification in distance as a result of applying the function to similar inputs. Low sensitive functions maps nearby inputs to the nearby outputs.

As mentioned before, differential privacy is compositional and queries can always be composed sequentially. In order to improve utilisation and reduce the privacy cost of analyses, *parallel queries* are introduced. Parallel queries are executed on disjoint subsets of records and provide a better lower bound on the privacy guarantee. Assume any arbitrary n queries, $Q_i, (1 \leq i \leq n)$ that each provide ϵ_i-differential privacy in isolation. If these queries are executed on n disjoint subsets of data, the lower bound for total privacy cost decreases from $\sum_{1 \leq i \leq n} \epsilon_i$ to $\max(\epsilon_1 \dots \epsilon_n)$. Parallel queries are especially useful for building histograms in which data points are grouped based on one of the attributes.

wPINQ [37] is a generalisation of PINQ to weighted datasets, capable of answering a larger class of queries like analyses on graphs. Since the amount of noise is significant, considering the fact that the input dataset is not usually the worst case, wPINQ assigns real valued weights to individuals and scales down the contribution of problematic individuals. This avoids high noise magnitude that is added as the result of applying high sensitive functions.

Airavat [39] is the integration of Mandatory Access Control (MAC) to differential privacy in order to provide confidentiality, integrity and privacy on large scale distributed cloud computing. Airavat prevents information leakage by using SELinux-like mandatory access control on the file system, modifying Java virtual machine and enforce differential privacy on its Map-Reduce framework.

Fuzz, DFuzz, Adaptive Fuzz [23, 38, 50] presents a typed functional programming language with a type system equipped with extra metrics to track function sensitivity to find the right amount of noise needed to guarantee differential privacy. Dfuzz [38] introduces dependent types as an extension to Fuzz, allowing a larger class of analyses; among them are analyses that need runtime information.

Fuzz also introduces some practical attacks using time and state side channels. Some precautions are introduced in PINQ to examine and rewrite methods that leak information via side channels. Despite their efforts, Fuzz exhibited their ineffectiveness. An analysis that observes an embarrassing record can run sub queries to use up the entire privacy budget. The exception caused by lack of budget can reveal the existence of the target record. The type system in this functional language infers the privacy cost *statically* which closes the privacy budget side channel. More importantly they introduced predictable transactions (micro queries) that process a single record in the fixed amount of time. They proved that the timing channel can be ruled out by aborting the micro query and replacing the result with a default value provided by the user.

Finally, they proposed network communication to separate the analyser and the machine running the query in order to eliminate electromagnetic radiation, power consumption and cache base side channels.

Relational Algebra [SQL] Relational algebra is commonly used as a theoretical foundation of query languages (like SQL) used to query relational databases. Palamidessi and Stronati [36] analysed and assigned tight sensitivity bounds to relational algebra operators and introduced composition rules that determine how sensitivity of a query can be computed from the sensitivity of the relational operator that has constructed it.

FLEX [25] is a practical tool for enforcing differential privacy for real world SQL queries based on elastic sensitivity. The elastic sensitivity is an upper bound on local sensitivity computed from the database metrics and the query structure.

GUPT [32] Noise magnitude can be adapted by considering the database that the query runs over rather than just the function logic that operates on it. This is achievable, as Nissim et al. [34] stated, by defining smooth sensitivity that represents the variability of a function in the neighbourhood of a particular instance of the database. This is challenging as the noise magnitude may leak information about the content of database and the accuracy of answers depends on the content of dataset. GUPT is a system that demonstrates this instance-based additive noise method. Computing or approximating smooth sensitivity might be difficult. GUPT uses a sample and aggregate framework to tackle this problem. This method can be automated and is applicable for an interactive model in which the function is given as a black-box.

IO Automata Tschantz et al. [47] introduce a formal probabilistic IO automaton model for differential privacy analyses. They use probabilistic bisimulation techniques to verify differential privacy of an interactive system inspired by PINQ. In addition to the proof technique, they show how bounded memory of their system causes increase in the privacy leakage bound.

CertiPriv Barthe et al. [5] introduce the *CertiPriv* framework that assists deriving the differential privacy guarantee provided by a program using *Probabilistic Relational Hoare Logic*. It is built on top of *Coq* proof assistant [46] and is capable of reasoning about approximate differential privacy. The method is powerful enough to build proofs of differential privacy for non-trivial mechanisms. Finally, they illustrate the generality of their approach by providing a correctness proof for several differential privacy mechanisms and algorithms.

RAPPOR [19] One real world tool that uses differential privacy is *Google Chrome* that anonymises users' browser preferences before sending reports to Google servers. Building on top of randomized responses,

RAPPOR sends software reports that look like random data, but can eventually in aggregation, be used to extract useful information. The generated report is only usable to gain statistical information about a user's client software setting without the possibility of inferring the exact user preferences.

Using primitive principles of differential privacy, it is easy to see how anonymisation of one single bit of response is applicable for anonymising bit strings. Taking the advantages of this, numerical or ordinal values can be represented by predicates on different ranges. A novel method of this system is the use of bloom filters to randomize non-categorical domains, like an arbitrary set of strings in a differentially private manner.

PROCHLO [6] is the implementation of the ESA (Encode, Shuffle, Analyze) architecture. The encoding is performed locally to transform, fragment and apply random noise to users' data. The shuffler strips metadata such as IP addresses, timestamps and further uses sampling and reordering of items to de-associate users from reports. Eventually the analyser decodes, aggregates and finally publishes the result.

DPCOMP [24] is a tool that allows researchers to evaluate different algorithms on a collection of datasets. Different factors that are influencing the performance of these algorithms are varied to measure their impacts on the accuracy and utility of the results.

Apple iOS 10 Apple introduced usage of differential privacy in their 10th edition of their operating system (iOS 10) in 2017 [3]. Several patents [44, 45] are registered explaining the details of the algorithms.

6 Contributions

This thesis comprises five papers, "*Featherweight PINQ*" [15], "*Differential Privacy, now it is getting personal*" [17], "*Sampling and Partitioning for Differential Privacy*" [18], "*PreTPost: A Transparent, User Verifiable, Local Differential Privacy Framework*" [16], "*Design and Use of PreTPost Framework*" [13].

Paper I, "*Featherweight PINQ*" published in the Journal of Privacy and Confidentiality, tries to fill the gap between the basic idea of differential privacy and PINQ, the standard and generic framework that implements differential privacy.

Many differential privacy frameworks share the same principles as PINQ. Looking closely at PINQ, one can see some of its deviations from the standard definition of differential privacy. We take a look at the concept of *privacy budget* and how PINQ *dynamically enforces* it, *parallel queries*, *strategies* and the program reflections upon query response. We present Featherweight PINQ, a simplified formal model of PINQ, that explains how PINQ works and proves correctness of the system.

Paper II, "*Personalised Differential Privacy, now it is getting personal*", is published in POPL[4] 2015. The paper tries to improve the budget book-keeping system that is used to *dynamically enforce* ϵ-differential privacy. In this novel method, rather than specifying a global privacy budget for a database, budget assignment is done on user level, hence personalised. Provenance tracking is used to compute the effect of query execution on an individual's budget. We model provenance tracking used in this method and show how this method reduces the privacy cost of analyses.

Paper III, *Sampling and Partitioning for Differential Privacy*, published in the fourteenth annual conference on "Privacy, Security and Trust 2016", looks into to two important basic blocks for constructing a differentially private analysis and their composition principles with existing blocks by introducing a new concept called probabilistic stability.

Paper IV, "*PreTPost: A Transparent, User Verifiable, Local Differential Privacy Framework*", shifts the perspective from a centralised setting to local differential privacy.

Finally the paper V, "*Design and Use of PreTPost Framework*", dives into practical aspects of using the PreTPost framework and the design decisions that are made to serve the framework's requirements.

In the remainder of this section we provide a more in-depth summary of these papers. Papers 1-3 are previously published and presented here unchanged modulo minor typographic corrections and reformatting to fit the style of the thesis.

6.1 Paper I : Featherweight PINQ

PINQ (Privacy Integrated Queries) [31], is one of the well-known tools for constructing differential privacy analyses. PINQ, like LINQ[5], brings the support for various data sources (e.g. DryadLINQ) and usage of general-purpose programming languages to write programs. These features make PINQ an appealing choice for analysers to construct programs that automatically enforce a certain level of differential privacy. The simplicity and extensibility of PINQ empowers experts to introduce variants of differential private frameworks. Among the frameworks that root in PINQ we can list wPINQ [37] for weighted datasets, Streaming-PINQ [49] for streaming algorithms and our tool ProPer for personalised differential privacy (introduced in Paper II).

Many articles have already discussed the theory, compositionality properties and the design of differentially private operations. However, formal verification of an implemented system, like PINQ, requires combining all these blocks and has not been the subject of any of those

[4] 42nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages

[5] Language-Integrated Query integrates query capabilities into the C# language.

studies. In addition, PINQ varies from differential privacy theories by introducing some extra features, the correctness of which have not been thoroughly investigated. In what follows we introduce the three aspects that PINQ handles but which have not been precisely introduced by the theory.

Partitioning Transformation PINQ introduces *parallel queries* by partitioning an input dataset into disjoint subsets. As a result, a parallel query enjoys the maximum of privacy guarantees of all sub-queries, which is lower than the sum of them. This can be explained as one individual from input dataset cannot be present in two disjoint partitions and a query on one partition does not reveal anything about other individuals that are present in the other partitions. This is not correct as the partitioning may also be done in an intermediate dataset rather than the input dataset. Information deriving from one individual may end up in two partitions, since an intermediate dataset may contain two records deriving from one individual.

Strategy for Sequential Queries A differential private program in this framework may inspect the result from one query and determine the parameters of the next query.

Privacy Budget PINQ dynamically enforces a privacy budget for analyses over a sensitive dataset by blocking any query that tries to over-consume the limited privacy budget. The theory only discusses the measuring of privacy cost, whereas PINQ enforces this privacy budget by throwing an exception when the budget is fully exhausted. Observing such behaviour may, in principle, be informative and leak information.

In Paper I, "Featherweight PINQ", we start by introducing the basic foundation of differential privacy that PINQ is based upon, such as ϵ-differentially aggregation operations, stable transformations and their composition with these aggregation operations to construct a query. Then we see how multiple queries are composed (sequential and parallel) and how it affects the overall privacy guarantee that the system promises. Finally, we analyse the gap between the theory and PINQ introduced as a result of:

- dynamically enforcing a certain level of privacy rather than measuring it
- inadequate modelling of adeptness of sequential and parallel queries
- execution of parallel queries on an intermediate table constructed as a result of table transformation

We present a model that simplifies the system in one important aspect, queries that set up a parallel query should be all present in the system on query execution time. To prove differential privacy of Featherweight PINQ, we introduce a probabilistic trace semantic. Using the

model and the trace semantics, we prove that any client program constructed in Featherweight PINQ yields a system satisfying differential privacy. Finally, we suggest two additional safe APIs that allow programs to read the global privacy budget value and the scaling factors for protected tables.

6.2 Paper II : Differential Privacy, Now it is Getting Personal

An important challenge is to improve analyses to gain more information without endangering an individual's information. However, much of the research up to now has tended to focus on improving mechanisms or specific algorithms rather than monitoring what parts of a database are leaked. If an analyst asks about the average salary of men, no information about women in the database is leaked.

Paper II tries to resolves these issues by introducing a new budget book-keeping method. In this setting each individual has a personalised privacy budget. Unlike PINQ in which the sensitivity of a function determines the multiplier of budget reduction from the global privacy budget, in the personalised setting the number of records derived from an individual participating in an analysis determines the multiplier of reduction from this personalised privacy budget. To keep track of records derived from one individual record we use a notion similar to *why provenance* in the terminology of [7]. The significance of the method can be highlighted in the following three points.

Efficient Budget Consumption Parallel queries are introduced in PINQ to reduce the cost of queries on global budget, but constructing parallel queries is cumbersome, because it requires presence of all queries on different partitions at the query execution time. Without the presence of all queries, finding a set of queries that optimises budget cost is not easy. In many cases the immediate answer is needed or there is some dependency between queries.

Good utility is one benefit that is gained from our personalised budget book-keeping technique. Unlike the classic model of budgeting, in which parallel queries play an important role for gaining proper utilization, this method does not rely on parallel queries since the system inherently enjoys efficient budget consumption. This approach also avoids many of the problems (mentioned in the previous section) in modelling parallel queries and designing parallel version of analyses. Assigning ϵ-personalized privacy budget to all individuals guarantees ϵ-differential privacy in the classic model.

Stream Data An added benefit of a personalised budget is in dynamic databases. In a dynamic setting, records may be added or removed from a working dataset continuously. Assigning one global privacy budget, that is decreased as queries are executed, implies that updating the entire database with fresh records does not affect the budget. This is unfortu-

nate since nothing has been leaked about these recent records and one may expect a non-zero privacy budget.

If a privacy budget is assigned to an individual's record when it is added to the dataset and the record can actively participate in analyses until its budget is exhausted, this model of budgeting seems to perfectly fit the requirements of stream databases. We show that our system is capable of answering continuous queries when the data-stream is refreshing the database at a certain rate.

Limitation and Disadvantages Behaviour of the system in corner cases in which particular records participate in more analyses is a controversial case. PINQ blocks a query at the execution time when the global privacy budget is exhausted and this behaviour of PINQ is shown not to leak information. In contrast, the same behaviour in the personalised setting shows the presence of individuals and is considered extremely informative. As an example, if the system blocks a query that intentionally targeted one individual, this clearly indicates that the individual is present in the database. Therefore, this method excludes records that are derived from individuals who do not have a sufficient budget to participate in analyses. Although this makes reasoning about utilization difficult, an analyser with some background information can choose and arrange queries in such a way that error from the absence of depleted records has an insignificant effect on the result.

We present our system, named *ProPer* (Provenance for Personalised Differential Privacy) that is modelled using a similar trace semantics as explained in Paper II with provenance tracking to find track records that are derived from individuals in the input dataset.

Provenance tracking adds overhead to transformation and query execution but comparing execution time of some analyses written in both PINQ and ProPer we demonstrate that the runtime overhead of provenance tracking is modest.

6.3 Paper III : Sampling and Partitioning for Differential Privacy

This paper studies sampling and partitioning methods and their effect on differential privacy in the unbounded differential privacy. We demonstrate a practical attack on PINQ's deterministic sampling transformation. The attack gives an example of when sampling has a negative effect on the differential privacy cost of an analysis compared to the case without sampling.

Furthermore, we investigated other sampling and partitioning methods, especially those that use randomness in their selection and compare them with their deterministic alternatives. To compare and use our result in differential private frameworks such as PINQ, we introduce the concept of probabilistic stability that generalises stability.

We made the interesting observation that only Bernoulli sampling boosts privacy and the partitioning method based on Bernoulli sampling has the same effect as the deterministic parallel composition.

6.4 Paper IV : PreTPost: A Transparent, User Verifiable, Local Differential Privacy Framework

Trust is an important factor in the privacy discussions. In all the previous papers, a trusted party possesses all the data and an analyst or a protection mechanism is responsible to ensure the correct anonymisation of the results. This model, commonly referred to as the centralised model, additionally requires confidentiality for data transformation and security for data storage from both internal and external attackers while the data is stored in the database. Many people do not trust the data curator or see the risk of failure as not acceptable.

The recent adaptation of a local model in the Chrome web browser [19] and the iOS operation system [3] seems to be a step forward toward respecting users' privacy but lack of transparency puts all the efforts in the question. Tang et al. [43] show that the implementation is promiscuous in the choice of the privacy parameter epsilon. Analyses in these systems are either hard-coded in the system or should be communicated to the users. Hard-coded analysis reduces the flexibility while dynamic analysis may endanger the security and confidentiality of the user data.

The analysis that is sent and executed on user data may introduce new security and privacy concerns and makes the need for transparency in the design more severe. These shortcomings bring the idea of a framework with dynamic analysis with transparent design in which the users verify the differential privacy of an analysis themselves with no need to trust the curator.

Pre-processing of the private data in the centralised model requires a complex sensitivity analysis of the algorithm. The post-processing that is common between the local and the centralised setting with the pre-processing theorems in the local model are simple but important fundamental concepts that the PreTPost framework is built upon.

The decomposition of analyses into *Pre,* T and *Post* components and restricting their communication simplifies their isolation and helps the system to block the leakage of information from timing and other types of side channels.

The usefulness of the system is demonstrated by integrating existing local differential privacy algorithms in the PreTPost framework. Additionally, one specific algorithm that is a core to several other algorithms is studied in detail to explain the decomposition technique.

6.5 Paper V : Design and Use of PreTPost Framework

Last section is a tutorial paper that discussed the use and the design of the PreTPost framework. The PreTPost tries to push back the trust

from the analyst who performs analyses to the transparent system that execute the analyses. Therefore, knowing the structure, architecture and the decisions that are made during the design and implementation of the PreTPost framework is important.

Local model tries to minimize the data storage in the curator side therefore the steady communication between users and the curator is important. We see how users can encode their privacy preference as a policy and how the curator and user agree on utility and privacy.

Additionally, we show how the framework can be deployed in a real scenario. We use a scenario of a Internet Service Provider (ISP) who is interested to learn about its client by querying their home routers. In these examples we demonstrate the decomposition of an analysis into pre-processing, randomised transformation and post-processing and present code samples for constructing a query and transmitting it to users.

Finally, we investigate threats from a malicious curator who is trying to extract user's information through unconventional methods. The pre/post processing functions can be a binary executable that may potentially be malicious. We look into the required isolation and sandboxing of different components of the system and how the system overcomes the information leakage through side channels (e.g. timing channel).

6.6 Personal Contributions

"*Featherweight PINQ*" [15] is the simple model we provided for PINQ. The proof of differential privacy for Featherweight PINQ was done by me and the development of the idea and system modelling was jointly done with David Sands.

During my master thesis project, I came up with the idea of a personalised budget and implemented a tool named *PINQuin* [14] to demonstrate it. ProPer (the implementation of which has some minor differences with the original idea of PINQuin) is a powerful model for a system that does provenance tracking in order to dynamically determine who and how much to subtract from the personalised budget. Details of this method are articulated in *Differential Privacy, Now it is Getting Personal* [17]. Modelling and the proof of ProPer were jointly done with David Sands with the help and contribution of Gerardo Schneider.

The idea for the article "*Sampling and Partitioning for Differential Privacy*" [18] came up when I noticed a problem in the PINQ's `Take()` and `Skip()` methods. The proof was developed by me, corrected and improved by David Sands and Thibaud Antignac.

The paper "*PreTPost: A Transparent, User Verifiable, Local Differential Privacy Framework*" [16] describes the framework that is developed after similarities between local differential privacy algorithms are observed and it was decided to unify them in one single framework. The algorithm's integration into the framework by their decomposition into

Pre,T,Post components and the idea of pre-processing theorem were done by me.

Finally I contributed to at least 50% of writing task for each paper except the paper "*Design and Use of PreTPost Framework*" [13] that is fully done by myself.

References

[1] Mohammad Alaggan, Sébastien Gambs, and Anne-Marie Kermarrec. Heterogeneous differential privacy. In *Theory and practice of differential privacy (TPDP 2015) at London, UK*, 2015.

[2] Miguel E. Andrés, Nicolás E. Bordenabe, Konstantinos Chatzikokolakis, and Catuscia Palamidessi. Geo-indistinguishability: differential privacy for location-based systems. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, CCS '13. ACM, 2013.

[3] Apple Press Release. Apple previews ios 10, the biggest ios release ever. 2016.

[4] Michael Barbaro and Tom Zeller. A face is exposed for aol searcher no. 4417749. Technical report, The New York Times, 2006.

[5] Gilles Barthe, Boris Kópf, Federico Olmedo, and Santiago Zanella-Béguelin. Probabilistic relational reasoning for differential privacy. *ACM Trans. Program. Lang. Syst.*, 2013.

[6] Andrea Bittau, Úlfar Erlingsson, Petros Maniatis, Ilya Mironov, Ananth Raghunathan, David Lie, Mitch Rudominer, Usharree Kode, Julien Tinnés, and Bernhard Seefeld. Prochlo: Strong privacy for analytics in the crowd. *CoRR*, abs/1710.00901, 2017.

[7] James Cheney, Laura Chiticariu, and Wang-Chiew Tan. Provenance in databases: Why, how, and where. *Found. Trends databases*, 1(4): 379–474, April 2009.

[8] Cynthia Dwork. A firm foundation for private data analysis. *Communications of the ACM*, 54(1):86–95, 2011.

[9] Cynthia Dwork, Krishnaram Kenthapadi, Frank McSherry, Ilya Mironov, and Moni Naor. Our data, ourselves: Privacy via distributed noise generation. In *Proceedings of the 24th Annual International Conference on The Theory and Applications of Cryptographic Techniques*, EUROCRYPT'06, pages 486–503. Springer-Verlag, 2006.

[10] 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.

[11] Cynthia Dwork, Moni Naor, Toniann Pitassi, and Guy N. Rothblum. Differential privacy under continual observation. In *Proceedings of the Forty-second ACM Symposium on Theory of Computing*, STOC '10, pages 715–724, New York, NY, USA, 2010. ACM.

[12] Cynthia Dwork, Moni Naor, Toniann Pitassi, Guy N Rothblum, and Sergey Yekhanin. Pan-private streaming algorithms. In *ICS*, pages 66–80, 2010.

[13] Hamid Ebadi. Design and use of pretpost framework. 2018.

[14] Hamid Ebadi. PINQuin, a framework for differentially private analysis. Master's thesis, Chalmers University of Technology, 2013.

[15] Hamid Ebadi and David Sands. Featherweight PINQ. *arXiv preprint arXiv:1505.02642*, 2012.

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

[17] Hamid Ebadi, David Sands, and Gerardo Schneider. Differential privacy: Now it's getting personal. In *Proceedings of the 42Nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '15. ACM, 2015.

[18] Hamid Ebadi, Thibaud Antignac, and David Sands. Sampling and partitioning for differential privacy. In *14th Annual Conference on Privacy, Security and Trust*. IEEE, 2016.

[19] Úlfar Erlingsson, Aleksandra Korolova, and Vasyl Pihur. RAPPOR: randomized aggregatable privacy-preserving ordinal response. *CoRR*, abs/1407.6981, 2014.

[20] Quan Geng and P. Viswanath. The optimal mechanism in differential privacy. In *Information Theory (ISIT), 2014 IEEE International Symposium on*, pages 2371–2375, June 2014.

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

[22] Bernard G. Greenberg, Abdel-Latif A. Abul-Ela, Walt R. Simmons, and Daniel G. Horvitz. The unrelated question randomized response model: Theoretical framework. *Journal of the American Statistical Association*, 64(326):pp. 520–539, 1969.

[23] Andreas Haeberlen, Benjamin C. Pierce, and Arjun Narayan. Differential privacy under fire. In *Proceedings of the 20th USENIX Security Symposium*, 2011.

[24] Michael Hay, Ashwin Machanavajjhala, Gerome Miklau, Yan Chen, Dan Zhang, and George Bissias. Exploring privacy-accuracy tradeoffs using dpcomp. In *Proceedings of the 2016 International Conference on Management of Data*, SIGMOD '16, pages 2101–2104, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-3531-7.

[25] Noah M. Johnson, Joseph P. Near, and Dawn Xiaodong Song. Practical differential privacy for SQL queries using elastic sensitivity. *CoRR*, abs/1706.09479, 2017.

[26] Zach Jorgensen, Ting Yu, and Graham Cormode. Conservative or liberal? personalized differential privacy. In *International Conference on Data Engineering (ICDE)*, 2015.

[27] Daniel Kifer and Ashwin Machanavajjhala. No free lunch in data privacy. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data*, SIGMOD '11, pages 193–204, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0661-4.

[28] Michal Kosinski, David Stillwell, and Thore Graepel. Private traits and attributes are predictable from digital records of human behavior. *Proceedings of the National Academy of Sciences*, 110(15): 5802–5805, 2013.

[29] Michal Kosinski, Yoram Bachrach, Pushmeet Kohli, David Stillwell, and Thore Graepel. Manifestations of user personality in website choice and behaviour on online social networks. *Machine Learning*, 95(3):357–380, Jun 2014. ISSN 1573-0565.

[30] Frank McSherry and Kunal Talwar. Mechanism design via differential privacy. In *Proceedings of the 48th Annual IEEE Symposium on Foundations of Computer Science*, FOCS '07, pages 94–103. IEEE Computer Society, 2007.

[31] Frank D McSherry. Privacy integrated queries: an extensible platform for privacy-preserving data analysis. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*, pages 19–30. ACM, 2009.

[32] Prashanth Mohan, Abhradeep Thakurta, Elaine Shi, Dawn Song, and David Culler. Gupt: privacy preserving data analysis made easy. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, pages 349–360. ACM, 2012.

[33] Arvind Narayanan and Vitaly Shmatikov. Robust de-anonymization of large sparse datasets. In *Proceedings of the 2008 IEEE Symposium on Security and Privacy*, SP '08, pages 111–125, Washington, DC, USA, 2008. IEEE Computer Society. ISBN 978-0-7695-3168-7.

[34] Kobbi Nissim, Sofya Raskhodnikova, and Adam Smith. Smooth sensitivity and sampling in private data analysis. In *Proceedings of the Thirty-ninth Annual ACM Symposium on Theory of Computing*, STOC '07, pages 75–84, New York, NY, USA, 2007. ACM.

[35] Kobbi Nissim, Thomas Steinke, Alexandra Wood, Micah Altman, Aaron Bembenek, Mark Bun, Marco Gaboardi, David O'Brien, and Salil Vadhan. Differential privacy: A primer for a non-technical audience (preliminary version), 2017.

[36] Catuscia Palamidessi and Marco Stronati. Differential privacy for relational algebra: Improving the sensitivity bounds via constraint systems. In *Proceedings 10th Workshop on Quantitative Aspects of Programming Languages and Systems*, 2012.

[37] Davide Proserpio, Sharon Goldberg, and Frank McSherry. A workflow for differentially-private graph synthesis. *CoRR*, abs/1203.3453, 2012.

[38] Jason Reed and Benjamin C. Pierce. Distance makes the types grow stronger: A calculus for differential privacy. In *Proceedings of the*

15th ACM SIGPLAN International Conference on Functional Programming, ICFP '10, pages 157–168, 2010.

[39] Indrajit Roy, Srinath T. V. Setty, Ann Kilzer, Vitaly Shmatikov, and Emmett Witchel. Airavat: Security and privacy for mapreduce. In *NSDI*, pages 297–312. USENIX Association, 2010.

[40] Rebecca Skloot. The immortal life of henrietta lacks, the sequel. Technical report, The New York Times, 2013.

[41] Thomas Steinke and Jonathan Ullman. Between pure and approximate differential privacy. *CoRR*, abs/1501.06095, 2015.

[42] Latanya Sweeney. K-anonymity: A model for protecting privacy. *Int. J. Uncertain. Fuzziness Knowl.-Based Syst.*, 10(5):557–570, 2002.

[43] Jun Tang, Aleksandra Korolova, Xiaolong Bai, Xueqiang Wang, and XiaoFeng Wang. Privacy loss in apple's implementation of differential privacy on macos 10.12. *CoRR*, abs/1709.02753, 2017.

[44] A.G. Thakurta, A.H. Vyrros, U.S. Vaishampayan, G. Kapoor, J. Freudiger, V.R. Sridhar, and D. Davidson. Learning new words, 2017. US Patent 9,594,741.

[45] A.G. Thakurta, A.H. Vyrros, U.S. Vaishampayan, G. Kapoor, J. Freudinger, V.V. Prakash, A. Legendre, and S. Duplinsky. Emoji frequency detection and deep link frequency, July 11 2017. US Patent 9,705,908.

[46] The Coq Development Team. The coq proof assistant reference manual, 2015. http://coq.inria.fr.

[47] Michael Carl Tschantz, Dilsun Kaynar, and Anupam Datta. Formal verification of differential privacy for interactive systems (extended abstract). *Electron. Notes Theor. Comput. Sci.*, 276:61–79, September 2011.

[48] Stanley L. Warner. Randomized response: A survey technique for eliminating evasive answer bias. *Journal of the American Statistical Association*, 60(309):pp. 63–69, 1965.

[49] Lucas Waye. Privacy integrated data stream queries. In *Proceedings of the 5th annual conference on Systems, programming, and applications: software for humanity*. ACM, 2014.

[50] Daniel Winograd-Cort, Andreas Haeberlen, Aaron Roth, and Benjamin C. Pierce. A framework for adaptive differential privacy. *Proc. ACM Program. Lang.*, 1(ICFP):10:1–10:29, August 2017. ISSN 2475-1421.

Paper I : Featherweight PINQ

Journal of Privacy and Confidentiality: Volume 7 : Issue 2, Article 7.

Erratum

We mistakenly claimed (p31, first item in the list) that the paper *"Privacy Integrated Queries: An Extensible Platform for Privacy-preserving Data Analysis"* [9] misses the adaptiveness argument:

> "Sequential composition is described in an oversimplified way, assuming that the queries are chosen independently from each other. ... To argue correctness this adaptiveness should be modelled explicitly."

Although the statement of the theorem does not mention adaptivity, in the conference version of the article it is clear from the surrounding text that adaptivity to the results of previous queries is taken into account. We thank Frank McSherry for pointing out our error.

Featherweight PINQ

Hamid Ebadi and David Sands

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

Abstract. Differentially private mechanisms enjoy a variety of composition properties. Leveraging these, McSherry introduced PINQ (SIGMOD 2009), a system empowering non-experts to construct new differentially private analyses. PINQ is an LINQ-like API which provides automatic privacy guarantees for all programs which use it to mediate sensitive data manipulation. In this work we introduce *featherweight PINQ*, a formal model capturing the essence of PINQ. We prove that any program interacting with featherweight PINQ's API is differentially private.

Keywords: Differential privacy, dynamic database, PINQ, Formalization

1 Introduction

Differential privacy [3, 5, 4] shows that by adding the right amount of noise to statistical queries, one can get useful results, and at the same time provide a quantifiable notion of privacy. The definition of differential privacy for a query mechanism (a randomized algorithm) is made by comparing the results of a query on any database with or without any one individual: a query Q is ε-differentially private if the difference in probability of any query outcome on a data-set only changes by a factor of e^{ε} (approximately $1 + \varepsilon$ for small ε) whenever an individual is added or removed.

Of the many of papers on differential privacy, a mere handful (at the time of writing) describe implemented systems which provide more than just a static collection of differentially private operations. The first such system is the PINQ system of McSherry [10]. PINQ is designed to allow non-experts in differential-privacy to build privacy-preserving data analyses. The system works by leveraging a fixed collection of differentially private data aggregation functions (counts, averages, etc.), and a collection of data manipulation operations, all embedded with a LINQ-like interface from otherwise arbitrary C# code. PINQ mediates all accesses to sensitive data in order to keep track of the sensitivity of various computed objects, and to ensure that the intended privacy budget ε is not exceeded; a budget could be exceeded by answering too many queries with too high accuracy. In this way PINQ is intended to make sure that the analyst (programmer) does not inadvertently break differential privacy.

Foundations of PINQ McSherry argues the correctness of PINQ by pointing out the foundations upon which PINQ rests. In essence these are:

1. A predefined collection of aggregation operations (queries) on tables, each with a parameter specifying the required degree of differential privacy. Standard aggregation operations such as (noisy) count and average are implemented. The core assumption is that each aggregation operation Q with noise parameter ε, written here as Q_ε, is an ε-differentially private randomised function.
2. Sequential composition principle: if two queries performed in sequence (e.g. with differential privacy ε_1 and ε_2 respectively) then the overall level of differential privacy is safely estimated by summing the privacy costs of the individual queries $(\varepsilon_1 + \varepsilon_2)$.
3. Parallel composition principle: if the data is partitioned into disjoint parts, and a different query is applied to each partition, then the overall level of differential privacy is safely estimated by taking the maximum of the costs of the individual queries.
4. Stability composition: the stability of a database transformation T is defined to be c if whenever you add n extra elements to the argument of T, the result of T changes by no more than $n \times c$ elements. If you

first transform a database by T, then query the result with an ε-private query, the privacy afforded by the composition of the two operations is safely approximated by $c \times \varepsilon$.

These "foundations" of PINQ provide an intuition about how and why PINQ works, but although a novel aim of PINQ was "providing formal end-to-end differential privacy guarantees under arbitrary use", the foundations are inadequate to build an end-to-end correctness argument since they fall short of describing number of PINQ features of potential relevance to the question of its differential privacy:

- Sequential composition is described in an oversimplified way, assuming that the queries are chosen independently from each other. In practice the second query of a sequence is issued by client code which has received the result of the first query. Thus the second query may depend on the outcome of the first. To argue correctness this adaptiveness should be modelled explicitly.
- Parallel queries partition data, but the data which is partitioned might not be the original input, but some intermediate table. The informal argument for taking the maximum of the privacy costs of the query on each partition relies on the respective queries applying to disjoint data points. But the data might not be disjoint when seen from the perspective of the original data set of individuals. Data derived from a participant might end up in more than one partition, so a correctness argument must model this possibility to show that it is safe.
- As for sequential composition, parallel queries are not parallel at all, but can be adaptive - the result of a query on one partition might depend on the result of a query on another. This means that the implementation greatly complicates the bookkeeping necessary to track the "maximum" cost of the queries.
- The foundations suggest how to compute the privacy cost of composed operations from their privacy and stability properties. But in practice PINQ does not *measure* the amount of privacy lost by a PINQ program, it *enforces* a stated bound. Because of this, there are two kinds of results from a query: the normal noisy answer, or an exception. An exception is thrown if answering the query normally would break the global privacy budget. To prove differential privacy it is not enough that the query is differentially private in the normal case – it must also be shown to be private in the case when an exception is thrown, since this information is communicated to the program.

In this paper we provide a foundation for PINQ by defining a minimalistic semantics, *Featherweight PINQ*, intended to model it's essence, while at the same time abstracting away from less relevant implementation details.

By idealising the interface we make clear the intended implementation, but not the details of its realisation in any particular language. Thus we model the client program completely abstractly as a deterministic labelled transition system which interacts with tables via the PINQ-like API but which is otherwise unconstrained. For this model we instantiate the definition of differential privacy, taking into account the interactive nature of the system, and prove that Featherweight PINQ provides differential privacy for any program.

2 PINQ

In this section we provide a brief description of the PINQ system from the user prespective. PINQ is a .NET API which provides an interface similar to the Language Integrated Queries (LINQ) that is a language extension to .NET. Analyses that use PINQ are typically written in C#.

Listing 3.1 shows a code fragment for a sample analysis producing the average ages of adult males and adult females, respectively and then separately compute the average of age for all individuals.

Listing 3.1. PINQ sample code

```
1  var agent = new PINQAgentBudget(budget);
2  var data = new PINQueryable<recordstype>(rawdata, agent);
3  var adults = data.Where(x => x.age > 17);
4  var genders = new [] {0,1};
5  var parts = adults.Partition(genders, x=>x.gender);
6  foreach (var a in genders) {
7     Console.WriteLine("Average age of {0} is {1}",
8                       a==0? "Males " : "Females ",
9                       parts[a].NoisyAverage(budget/2, x=>x.age)
10 }
11 Console.WriteLine("Average age (all):" + data.NoisyAverage(c))
```

The first two lines of the program initialises a PINQueryable object with sample sensitive data (`rawdata`) structures and set the privacy limit (`budget`). A PINQueryable object is a wrapper to the database which enables PINQ to track the properties that are relevant for differential privacy. The supplied "agent" parameter expresses the amount of differential privacy that the system will enforce on this database.

The analysis starts by selecting (line 3) a subset of records of interest (those who are adults). Behind the scenes PINQ records the fact that the *stability* of `data` is unchanged: adding a single record to the `rawdata` does not change the size of the result of this transformation by more than a single record.

In line 5 a partitioning operation splits the data into two groups based on the gender field (0 for Male, 1 for Female). Partition is not a standard LINQ/SQL style operation, but is specific to PINQ. For each partition (i.e. for each gender), the code outputs a noisy average of the `age`. `NoisyAverage` is one of a collection of built-in differentially private primitive aggregation operations provided by PINQ. The amount of differen-

tial privacy for each query in the loop is `budget/2`. After executing the `foreach` loop there will be `budget/2` of the original budget remaining. The outcome of the last line depends on the accuracy/privacy parameter `c`. If c is larger than `budget/2` the program will throw an exception (because answering the query with that degree of precision would break the budget).

3 Idealised Program

In this section we describe the abstract model of the program and API to the PINQ operations. In the section thereafter we go on to model the PINQ internals, what we call the *protected system*, before combining these components into a the overall model of Featherweight PINQ.

The first thing that we will abstract away from is the host programming language. Here one could chose to model a simple programming language, but it is not necessary to be that concrete. Instead we model a program as an arbitrary deterministic system that maintains its own internal state, and issues commands to the PINQ internals. In this sense we idealise PINQ by assuming that the API cannot be bypassed. In fact the PINQ system does not successfully encapsulate the all the protected parts of the system, and so some programs can violate differential privacy by bypassing the encapsulation [8], or by using side effects in places where side-effects are not intended. By idealising the interface we make clear the intended implementation, but not the details of its realisation in any particular language. By treating programs abstractly we also simplify other features of PINQ including aspects of its architecture which promote certain forms of extensibility.

Before describing the program model it is appropriate to say a few words about the *protected system* (described formally in the next section). The protected system contains all the datasets (tables) manipulated by the program. Since these are the privacy sensitive data, we only permit the program to access them via the API. The protected system tracks the stability of all the tables which it maintains, together with a global budget. Our program interacts with the protected system by the following operations:

Assignment Tables in the protected system are referred to via *table variables*. A program can issue an assignment command. The model allows the program to manipulate a table using transformation that assign a new value to table variables.

The general form of assignment is of the form
$tv := F(tv_1, \ldots, tv_n)$, where F is taken from a set of *function identifiers* representing a family of transformations with bounded stability (i.e. for each argument position i there is a natural number c_i such that if the size of the ith argument changes by n elements, then the result will change by at most $c_i \cdot n$ elements). This stability requirement comes from

PINQ and is discussed in more detail in the next section. Transformations include standard operations such as the `.Where(x => x.age > 17)` from the example in listing 3.1, and simple assignments $t_1 := t_2$ (taking F to be the identity function), as well as assignments of literal tables (the case when F has arity 0).

Query The only other operation of the PINQ API is the application of a primitive differentially private query. In the example above we saw a compound transformation and query operation:
`parts[a].NoisyAverage(budget/2, x=>x.age)` . It is sufficient to model just the query, since the transformation (`x=>x.age`) can be implemented via an intermediate assignment. Thus we assume a set of primitive queries **Query**, ranged over by Q, which take as argument a positive real (the ε parameter) and a table, and produce a discrete probability distribution over a domain of result values **Val**.

We generalise the single query operation to a *parallel query*, with syntax $\mathsf{query}(tv, f, \vec{Q}, \varepsilon)$, where

1. tv is the table variable referring to the table that will be used for the analysis,
2. f is the partitioning function that maps each record to an index in $\mathsf{codomain}(f) = \{1, \ldots, k\}$ for some $k \in \mathbb{N}$,
3. $\vec{Q}$ is a vector of k queries from **Query**.

The execution of this operation (as described in the next section) involves computing the sequence of randomised values

$$Q_i(\varepsilon, \{r \in T \mid f(r) = i\}), i \in \mathsf{codomain}(f)$$

where T is the table bound to tv. This is the "parallel query" operation described informally in the description of PINQ [10]. We use a single ε for all queries because if we chose an ε_i for each query the privacy cost will be maximum of all the epsilons in any case, so we may as well enjoy the accuracy of the largest epsilon. However, we note that the implementation of PINQ is more general than this, since the queries on each partition may be performed in an adaptive way. Here we are making a trade-off in keeping our model simple at the expense of not proving differential privacy for quite as general a system.

Client Program Model The above abstraction of the PINQ API allows us to abstract away from all internal details of the programming language using the API. Following [6] we model a program as an arbitrary labelled transition system with labels representing the API calls:

Definition 1 (ProgAct Labels). *The set of program action labels* **ProgAct***, ranged over by a and b, are defined as the union of three syntactic forms:*

1. *the distinguished action τ, representing computational progress without interaction with the protected system,*

2. $tvar := F(tv_1, \ldots, tv_n)$ *where* F *is a function identifier, i.e. the formal name of a transformation operation of arity* n,
3. $\mathsf{query}(tv, f, \vec{Q}, \varepsilon)?\vec{v}$, *where* f *is a function from records to* $\{1, \ldots, k\}$ *for some* $k > 0$, *where* $\vec{v}$ *is a vector of values in* $\mathbf{Val}^k$, *and* $\vec{Q}$ *is a vector of* k *queries.*

Every label represents an interaction between a client program and the protected system. The labels represent the observable output of a system which are a sequence of those actions: internal (silent) steps (τ) modelling no interaction, and vectors of values $\vec{v}$ which are the results of some query being answered and returned to the program.

To define these transitions, we assume a client program modelled by a labelled transition system modelling the API to the protected system. For client programs, the label corresponding to a query call is of the form $\mathsf{query}(tv, f, \vec{Q}, \varepsilon)\ ?\ \vec{v}$, and models the pair of query and the returned result (as described before) as a single event. This allows us to model value passing with no need to introduce any specific syntax for programs. Note that the value returned by the query is known to the program, and the program can act on it accordingly. From the perspective of the program and the protected system together, this value will be considered an observable output of the whole system.

Definition 2 (Client Program). *A* client program *is a labelled transition system* $\langle \mathbb{P}, \rightarrow, P_0 \rangle$, *with labels from* $ProgAct$, *where* $\mathbb{P}$ *is all possible program states,* P_0 *is the initial state of the program, and the transition relation* $\rightarrow \subseteq (\mathbb{P} \times \mathbf{ProgAct} \times \mathbb{P})$ *is deadlock-free, and satisfies the following determinacy property: for all states* P, *if* $P \xrightarrow{a} P'$ *and* $P \xrightarrow{b} P''$ *then*

1. *if* $a = b$ *then* $P' = P''$,
2. *if* a *is not a query then* $a = b$,
3. *if* $a = \mathsf{query}(tv, f, \vec{Q}, \varepsilon)\ ?\ \vec{v}$ *then* $b = \mathsf{query}(tv, f, \vec{Q}, \varepsilon)\ ?\ \vec{u}$ *for some* $\vec{u}$, *and for all actions* c *of the form* $\mathsf{query}(tv, f, \vec{Q}, \varepsilon)\ ?\ \vec{w}$ *there exists a state* P_c *such that* $P \xrightarrow{c} P_c$.

The conditions on client programs are mild. Deadlock (i.e. termination) freedom simplifies reasoning; a program that terminates in the conventional sense can be modelled by adding a transition $P \xrightarrow{\tau} P$ for all terminated states P. Query transitions model both the query sent and the result received. Since we are modelling message passing using just transition labels, the condition on queries states that the program must be able to accept any result from a given query. Modulo the results returned by a query, the conditions require the program to be deterministic. This is a technical simplification which (we believe) does not restrict the power of the attacker.

Remark: Implicit parameters We will prove that Featherweight PINQ provides differential privacy for any client program. To avoid excessive parametrisation of subsequent definitions, in what follows we will fix some arbitrary client program $\langle \mathbb{P}, \rightarrow, P_0 \rangle$ and some arbitrary initial budget ε and make definitions relative to these.

4 Featherweight PINQ

In this section we turn to the model of the internals of PINQ, and the overall semantics of the system. We begin by describing the components of the protected system, and then give the overall model of Featherweight PINQ by giving a probabilistic semantics (as a probabilistic labelled transition system) to the combination of a client program and a protected system.

4.1 The Protected System

Global Privacy Budget The first component of the protected system is the global privacy budget. This is a non-negative real number representing the remaining privacy budget. The idea is that if we begin with initial budget b then Featherweight PINQ will enforce b-differential privacy. The global budget is decremented as queries are computed, and queries are denied if they would cause the budget to become negative. In PINQ the budget is associated with a given data source. In our model we assume that there is only one data source, and hence only one budget. Further, PINQ allows the budget to be divided up and passed down to subcomputations. This does not fundamentally change the expressiveness of PINQ since, as we show later, we are free to extend Featherweight PINQ with the ability to query the global budget directly. Thus any particular strategy for dividing the global budget between subcomputations can be easily programmed.

The Table Environment The other data component of the protected system is the table environment, which maps each table variable to the table it denotes, together with a record of the *scaling factor*, which is a measure of the *stability* of the table relative to the initial data set. We define this precisely below. Formally we define a table as power-set of records, $\mathcal{P}(\mathbf{Record})$, a protected table is a pair of a table with its scaling factor:

$$\mathbf{ProtectedTable} \stackrel{\text{def}}{=} \mathbf{Table} \times \mathbb{N}$$

4.2 The Featherweight PINQ Transition System

Featherweight PINQ is defined by combining a client program with the protected system to form the states of a probabilistic transition system.

Definition 3 (Featherweight PINQ States). *The* states *(otherwise known as* configurations*) of Featherweight PINQ, ranged over by* $\mathbb{C}$*,* $\mathbb{C}'$ *etc., are triples of the form* $\langle P, E, B \rangle$ *where* P *is a client program state,*

$E \in \mathbf{TVar} \to \mathbf{ProtectedTable}$ *is the table environment, and* $B \in \mathbb{R}^+$ *is the global budget.*

There is a family of possible initial states, indexed by the distinguished input table, and the initial budget. We define these by assuming the existence of a distinguished table variable, *input*, which we initialise with the input table, while all other table variables are initialised with the empty table:

Definition 4 (Initial configuration).

$$\mathbf{Init}(T, B) \stackrel{\text{def}}{=} \langle P_0, E_T, B\rangle \text{ where } E_T(tv) \triangleq \begin{cases} (T, 1) & \text{if } tv = input \\ (\{\}, 0) & \text{otherwise.} \end{cases}$$

The operational semantics of featherweight PINQ can now be given:

Definition 5 (Semantics). *The operational semantics of configurations is given by a probabilistic labelled transition relation with transitions of the form* $\mathbb{C} \xrightarrow{a}_p \mathbb{C}'$ *where* $a \in \mathbf{Act} \stackrel{\text{def}}{=} \{\tau, \bot\} \cup \bigcup_{n\in\mathbb{N}} \mathbf{Val}^n$*, and (probability)* $p \in [0, 1]$*. The definition is given by cases in Figure 1.*

$$\text{Silent}\ \frac{P \xrightarrow{\tau} P'}{\langle P, E, B\rangle \xrightarrow{\tau}_1 \langle P', E, B\rangle}$$

$$\text{Assign}\ \frac{P \xrightarrow{tv:=F(tv_1,\ldots,tv_n)} P'}{\langle P, E, B\rangle \xrightarrow{\tau}_1 \langle P', E[tv \mapsto (T', s)], B\rangle}\ \text{where}\ \begin{cases} E(tv_i) = (T_i, s_i), \\ \qquad\qquad i \in \{1,\ldots,n\} \\ \mathsf{stability}(F) = (c_1,\ldots,c_n) \\ s = \sum_{i=1}^n c_i \times s_i \\ T' = [\![F]\!](T_1,\ldots,T_n) \end{cases}$$

$$\text{Query}_\bot\ \frac{P \xrightarrow{\mathsf{query}(tv,f,\vec{Q},\varepsilon)\ ?\ \bot} P'}{\langle P, E, B\rangle \xrightarrow{\bot}_1 \langle P', E, B\rangle}\ \text{where}\ \begin{cases} E(tv) = (T, s) \\ \varepsilon \cdot s > B \end{cases}$$

$$\text{Query}\ \frac{P \xrightarrow{\mathsf{query}(tv,f,\vec{Q},\varepsilon)\ ?\ \vec{v}} P'}{\langle P, E, B\rangle \xrightarrow{\vec{v}}_p \langle P', E, B - s \cdot \varepsilon\rangle}\ \text{where}\ \begin{cases} E(tv) = (T, s), \quad \varepsilon \cdot s \le B \\ \mathsf{codomain}(f) = \{1,\ldots,n\} \\ \qqu\qquad \vec{v} \in \mathbf{Val}^n \\ T_i = \{s \mid s \in T, f(s) = i\}, \\ \qqu\qquad i \in \{1,\ldots,n\} \\ p = \prod_{i=1}^n \Pr[Q_i(\varepsilon, T_i) = v_i] \end{cases}$$

Fig. 1. Operational semantics

We note at this point that some of the primitives have not yet been defined (e.g. stability in the Assign rule), and that the rules of the system do not, *a priori*, define a probabilistic transition system. We will elaborate these points in what follows. We begin by explaining the rules in turn.

Assign When a program issues an assignment command $tv := F(tv_1, \ldots, tv_n)$, the value of the stored table for tv is updated in the obvious way. We must also record the scaling factor of the table thus computed. The scaling factor is computed from the scaling factors of the tables for $tv_1, \ldots, tv_n$, and the *stability* of the transformation f. We assume a mapping $[\![\cdot]\!]$ from formal function identifiers F to the actual table transformation functions $[\![F]\!]$ of corresponding arity.

Definition 6. *A table transformation f of arity n has stability $(c_1, \ldots, c_n)$ if for all $i \in \{1, \ldots, n\}$, we have*

$$|f(T_1, \ldots, T_i, \ldots, T_n) \ominus f(T_1, \ldots, T_i', \ldots T_n)| \leq c_i \times |T_i \ominus T_i'|$$

This is the n-ary generalisation of McSherry's definition [10], and bounds the size change in a result in terms of the size change of its argument. This is made more explicit in the following:

Lemma 1. *If f has stability $(c_1, \ldots, c_n)$ then*

$$|f(T_1, \ldots, T_n) \ominus f(T_1', \ldots, T_n')| \leq \sum_i^n (c_i \times |T_i \ominus T_i'|)$$

Note that not all functions have a finite stability. An example of this is the database join operation (essentially the cartesian product); adding one new element to one argument will add k new elements to the result, where k is the size of the other argument. Thus there is no static bound on the number of elements that may be added. Thus PINQ (and hence Featherweight PINQ) supports only transformation operations which have a finite stability. Table 4.2 illustrates the stability of some of the transformations that are introduced in PINQ. The variant of the join operation, Join* deterministically produces bounded numbers of join elements. For the purpose of this paper we do not need to be specific about the transformations. We simply assume the existence of a function $\mathsf{stability}$ which soundly returns the stability of a function identifier, i.e., if $\mathsf{stability}(F) = (c_1, \ldots, c_n)$ then $[\![F]\!]$ has stability $(c_1, \ldots, c_n)$.

The transition rule for assignment in featherweight PINQ is thus

$$\frac{P \xrightarrow{tv := F(tv_1, \ldots, tv_n)} P'}{\langle P, E, B\rangle \xrightarrow{\tau}_1 \langle P', E[tv \mapsto (T', s)], B\rangle} \text{ where } \begin{cases} E(tv_i) = (T_i, s_i), \\ \qquad\qquad i \in \{1,\ldots,n\} \\ \mathsf{stability}(F) = (c_1,\ldots,c_n) \\ s = \sum_{i=1}^n c_i \times s_i \\ T' = [\![F]\!](T_1,\ldots,T_n) \end{cases}$$

The label on the rule τ says that nothing (other than computational progress) is observable from the execution of this computation step. The subscript 1 is the probability with which this step occurs.

Understanding the scaling factor Here we provide more intuition about the scaling factor calculations, and explain some differences between the PINQ implementation and the Featherweight PINQ model. As

Transformation	Stability
Select(T, *maper*)	(1)
Where(T, *predicate*)	(1)
GroupBy(T_1, *keyselector*)	(2)
Join*(T_1,T_2, n, m, *keyselector*$_1$, *keyselector*$_2$)	(n,m)
Intersect(T_1,T_2)	(1,1)
Union(T_1,T_2)	(1,1)
Partition(T, *keyselector*, *keysList*)	(1)

Table 1. Transformation stability

an example, suppose we have a computation of a series of tables A–G depicted in Figure 2.

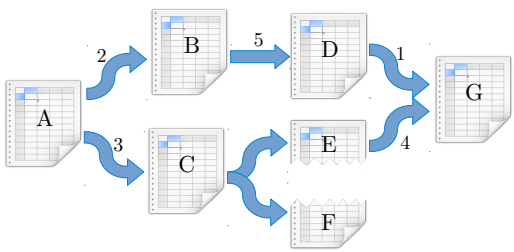

Fig. 2. Transformations

	s	Calculation
A	1	Input table
B	2	$s(A) \times 2$
C	1	$s(A) \times 3$
D	10	$s(B) \times 5$
E	3	$s(C)$
F	3	$s(C)$
G	22	$s(D) \times 1 + s(E) \times 4$

Fig. 3. Scaling factors (s)

The figure represents a PINQ computation involving three unary transformations (producing B, C, and D), one binary transformation producing G, and one partition operation (splitting C into E and F). We have labelled the transformation arcs with the stability constants of the respective transformations. What is the privacy cost of an ε differentially private query applied to, say, table D? Since D is the result of two transformations on the input data, the privacy cost is higher than just ε. The product of the sensitivities on the path from D backwards to the input A provide the *scaling factor* for ε. In this case the scaling factor for a query on D is 10. The remaining scaling factors are summarised in the table in Figure 3.

The scaling factor is the stability of that specific table; it bounds the maximum possible change of the table as a result of a change in the in-

put dataset, assuming that it was produced using the same sequence of operations. The scaling factor is computed from the stabilities of transformations that produced it. The scaling factor for each protected table (except input table which has the scaling factor one) is computed compositionally using the scaling factors (s_i) of all the arguments and the sensitivities of corresponding transformation arguments (c_i) using the following formula: $\mathrm{s}_A = \sum_{i \in parent(A)} \mathrm{c}_i \times \mathrm{s}_i$

Figures 2 and 3 allow us to explain two key differences between PINQ and our model:

1. In PINQ, the tree structure depicted in the figure is represented explicitly, and scaling factors are calculated lazily: at the point where a query with accuracy ε is made on a table it is necessary to calculate its scaling factor s in order to determine the privacy cost $s.\varepsilon$. To do this the tree is traversed from the query at the leaf back to the root, calculating the scaling factor along the way. At the root the total privacy cost is then known and deducted from the budget (providing the budget is sufficient). In Featherweight PINQ the scaling factors of each table are computed eagerly, so the tree structure is not traversed.
2. In Featherweight PINQ we restrict the partition operation to the leaves of the tree, and combine it with the application of primitive queries to partitions.

The consequence of these two simplifications is that we do not need to represent the PINQ computation tree at all – all computations are made locally at the point at which a table is produced or queried.

Queries Parallel queries were described in detail in the previous section. When a program issues a query is it represented as a parallel query and a possible result – i.e. we model the query and the returned result as a single step. There are two cases to consider, according to whether the budget is sufficient or not. If the queried table T has scaling factor s then the cost of an ε query is $s \times \varepsilon$. If this is greater than the current global budget then the result is the exceptional value $\perp$. This value is the observable result of the query, and it occurs with probability 1. On the other hand, if the budget is sufficient, then the vector of query results $\vec{v}$ is returned with probability $p = \prod_i \Pr[Q_i(\varepsilon, T_i) = v_i]$ where T_i is the ith partition of T. Note that p is indeed a probability, since the component queries are independent.

5 Differential Privacy for Featherweight PINQ

In this section we prove that Featherweight PINQ is differentially private. We begin by recapping the goals of differential privacy, before showing how to specialise the definition to Featherweight PINQ. Doing this entails building a trace semantics for Featherweight PINQ.

Differential privacy, guarantees that a data query mechanism (abstractly, a randomized algorithm) behaves similarly on similar input databases. This "similarity" is a quantitative measure ε on the difference in the information obtained from any data set with or without any individual. When this difference is small, the presence or absence of the individual in the data set is difficult to ascertain.

Definition 7. *Mechanism f provides ε-differential privacy if for any two datasets A and B that differ in one record ($| A \ominus B | = 1$), and for any two possible outcome $f(A)$ and $f(B)$, the following inequalities holds :* $e^{-\varepsilon} \leq \frac{\Pr[(f(A) \in S)]}{\Pr[(f(B) \in S)]} \leq e^{\varepsilon}$

In this definition, S is subset of the range of outcomes for f ($S \subseteq \text{Range}(f)$) and for similarity of outcomes we use the ratio between the probabilities of observing outcomes $\Pr[(f(datasets) \in S)]$ when the analyses are executed on any two similar datasets A and B. Finally for similarity of datasets hamming distance is used as a metric. In this work we assume that the primitive query mechanisms (and thus Featherweight PINQ) provide answers over a discrete probability distribution, so that it is sufficient to consider S to be a singleton set.

5.1 Trace semantics

The first step to instantiating the definition of differential privacy to Featherweight PINQ is to be able to view Featherweight PINQ as defining a probabilistic function. In fact each client program gives rise to a family of probabilistic functions, one for each length of computation that is observed. This is given by building a *trace semantics* on top of the transition system for Featherweight PINQ.

The semantics of Featherweight PINQ is a probabilistic labelled transition system of the simplest kind: for each configuration $\mathbb{C}$, the sum of all probabilities of all transitions of $\mathbb{C}$ is equal to 1. The system is also deterministic, in the sense that if $\mathbb{C} \xrightarrow{a}_{p_1} \mathbb{C}_1$ and $\mathbb{C} \xrightarrow{a}_{p_2} \mathbb{C}_2$ then $p_1 = p_2$ and $\mathbb{C}_1 = \mathbb{C}_2$. This makes it particularly easy to lift the probabilistic transition system from single actions to traces of actions:

Definition 8 (Trace semantics). *Define the trace transitions* $\Rightarrow \subseteq \textbf{Config} \times \textbf{Act}^* \times [0,1] \times \textbf{Config}$ *inductively as follows: (i)* $\mathbb{C} \overset{[]}{\Rightarrow}_1 \mathbb{C}$ *where* $[] \in \textbf{Act}^*$ *is the empty trace, and (ii) if* $\mathbb{C} \xrightarrow{a}_p \mathbb{C}'$ *and* $\mathbb{C}' \overset{t}{\Rightarrow}_q \mathbb{C}''$ *then* $\mathbb{C} \overset{at}{\Rightarrow}_{p.q} \mathbb{C}''$

Traces inherit determinacy from the single transitions:

Proposition 1 (Traces are Deterministic). *If* $\mathbb{C} \overset{t}{\Rightarrow}_{p_1} \mathbb{C}_1$ *and* $\mathbb{C} \overset{t}{\Rightarrow}_{p_2} \mathbb{C}_2$ *then* $p_1 = p_2$ *and* $\mathbb{C}_1 = \mathbb{C}_2$

This follows by an easy induction on the trace, using the fact that the single step transitions are similarly deterministic.

Lemma 2 (Traces are Probabilistic). *Define*

$$\mu(\mathbb{C}, t) \stackrel{\text{def}}{=} \begin{cases} p & \text{if } \mathbb{C} \stackrel{t}{\Rightarrow}_p \mathbb{C}' \\ 0 & \text{otherwise.} \end{cases}$$

For all configurations $\mathbb{C}$*, and all* $n > 0$*,*

$$\sum_{t \in Act^n} \mu(\mathbb{C}, t) \ = 1$$

where Act^n *is the set of traces of length* n*.*

The proof is a simple induction on n, using the proposition above. The lemma says that whenever $\mathbb{C} \stackrel{t}{\Rightarrow}_p$, then p is the probability that you see trace t after having observed $\mathsf{size}(t)$ steps of the computation of $\mathbb{C}$. We will thus refer to the probability of a given trace to mean the probability of producing that trace from the given configuration among all traces of the same length. We denote this by writing $\Pr[\mathbb{C} \stackrel{t}{\Rightarrow}] = p$ when $\mathbb{C} \stackrel{t}{\Rightarrow}_p$.

Differential Privacy for Traces We are now in a position to specialise the definition of differential privacy for Featherweight PINQ. How can we view Featherweight PINQ as a probabilistic function? The probabilistic function is determined by the client program (which we have kept implicit but unconstrained), the initial budget ε, and the length of trace n that is observed for any combination of these we define the function which maps a table T to trace t of length n with probability p precisely when $\Pr[\mathbf{Init}(T, \varepsilon) \stackrel{t}{\Rightarrow}] = p$.

The instantiation of the differential privacy condition to Featherweight PINQ is thus:

$$\forall t, T, T', \varepsilon. \text{ if } |T \ominus T'| = 1 \text{ then } \mathrm{e}^{-\varepsilon} \leq \frac{\Pr[\mathbf{Init}(T, \varepsilon) \stackrel{t}{\Rightarrow}]}{\Pr[\mathbf{Init}(T', \varepsilon) \stackrel{t}{\Rightarrow}]} \leq \mathrm{e}^{\varepsilon}$$

Towards a proof of this property we introduce some notation to reflect key invariants between the pairs of computations (for T and T' respectively).

Definition 9 (Similarity). *We define similarity relations* $\sim$ *between tables, environments, and configurations as follows:*

- *For tables* T *and* T'*, and* $s \in \mathbb{N}$ *define* $T \sim_s T'$ *("*T *is* s*-similar to* T'*") if and only if* $|T \ominus T'| \leq s$*.*
- *For protected environments* E *and* E'*, define* $E \sim E'$ *if and only if for all* tv*, if* $E(tv) = (T, s)$ *and* $E'(tv) = (T', s')$ *then* $s = s'$ *and* $T \sim_s T'$*.*
- *For configurations, define* $\langle P, E, B\rangle \sim \langle P', E', B'\rangle$ *if and only if* $P = P'$*,* $E \sim E'$ *and* $B = B'$*.*

The configuration similarity relation captures the key invariant between the two computations in our proof of differential privacy. First we need to show that the invariant is established for the initial configurations:

Lemma 3. *If* $T \sim_1 T'$ *then* $\mathbf{Init}(T, B) \sim \mathbf{Init}(T', B)$.

This follows easily form the definition of the initial configuration. Now the main theorem shows that this is maintained throughout the computation:

Theorem 1. *If* $T \sim_1 T'$ *and* $\mathbf{Init}(T, B) \stackrel{t}{\Rightarrow}_p \mathbb{C}$, *then* $\mathbf{Init}(T', B) \stackrel{t}{\Rightarrow}_q \mathbb{C}'$ *where* $\mathbb{C} \sim \mathbb{C}'$ *and* $p \leq q.\exp(B - \varepsilon)$ *for some* $\varepsilon \leq B$.

The proof, an induction over the length of the trace, is given in Appendix 1.

Corollary 1 (B-differential privacy). *If* $T \sim_1 T'$ *and* $\Pr[\mathbf{Init}(T, B) \stackrel{t}{\Rightarrow}] = p$ *then* $\Pr[\mathbf{Init}(T', B) \stackrel{t}{\Rightarrow}] = q$ *for some* q *such that* $p \leq q \cdot \exp(B)$.

6 Related Work

The approach described in this paper owes much to the model used in the formalisation developed in our recent work on *personalised differential privacy* [6]. The idea to model the client program as an abstract labelled transition system comes from that work. That work also shows how dynamic inputs can be handled without major difficulties.

The closest other prior work is developed by Tschantz et al [13]. Their work introduces a way to model interactive query mechanisms as a probabilistic automata, and develop bisimulation-based proof techniques for reasoning about the differential privacy of such systems. As a running example they consider a system "similar to PINQ", and use it to demonstrate their proof techniques. From our perspective their system is significantly different from PINQ in an number of ways: (i) it does not model the transformation of data at all, but only queries on unmodified input data, (ii) it models a system with a bounded amount of memory, and implements a mechanism which deletes data after it has been used for a fixed number of queries (neither of which relate to the implementation of PINQ). Regarding the proof techniques developed in [13], as previously noted in [6], a key difference between our formalisation and theirs is that they model a passive system which responds to external queries from the environment. In contrast, our model includes the adaptive adversary (the client program) as an explicit part of the configuration. In information-flow security (to which differential privacy is related) this difference in attacker models can be significant [15]. However it may be possible to prove that the passive model of [13] is sound for the active model described here (c.f. a similar result for interactive noninterference [2]).

Haeberlen et al [8] point out a number of flaws an covert channels in the PINQ system. This may seem at odds with our claims for the soundness of PINQ, but in fact all the flaws described are either covert timing channels (which we do not attempt to model), flaws in PINQ's

implementations of encapsulation, or failure to prevent unwanted side-effects, or combinations of these. Following this analysis, Haeberlen et al introduce a completely different approach to programming with differential privacy (an approach further developed and refined in [12] [7]) based on statically tracking sensitivity through sensitivity-types. This non-interactive approach is rigorously formalised and proven to provide differential privacy.

Barthe et al [1] introduce a relational Hoare-logic for reasoning formally about the differential privacy of algorithms. They include theorems relating to sequential and parallel composition of queries in the style of those stated by McSherry [10]. Unlike the present work, [1] does not rely on differentially private primitives, but is able to prove differential privacy from first principles.

7 Limitation and Extension

In this section we discuss what we see as the main limitations of Featherweight PINQ in relation to the PINQ system. We also discuss some easy extensions that become apparent from the proof of differential privacy.

Parallel Queries The form of parallel query that we model matches the informal description in [10], but is not as general as the construct found in the implementation. We believe that this is the main shortcoming in the Featherweight PINQ model, as more general form is interesting, and thus its correctness is not immediately obvious. (Whether the shortcoming has any practical significance in the way one might write programs is less clear.) The difference was described in Section 4 in connection with Figure 2, which depicts a partition operation which is not supported by Featherweight PINQ since it is not immediately followed by queries on the partitions. In fact the queries in PINQ need not be parallel at all, but can be adaptive (i.e., a query on one partition can be used to influence the choice of query on other partitions). This change is not easily supported by a small change to our model since it does not seem to be implementable using Featherweight PINQ's simple history-free use of explicit scaling factors. A proof of differential privacy for a more general protected system model encompassing this is left for future work.

Extensions to the PINQ API We mention one extension to PINQ that emerges from the details of the correctness proof. In PINQ, the budget and the actual privacy cost of executing an ε differentially private query on some intermediate table is not directly visible to the program:

> *"An analyst using PINQ is uncertain whether any request will be accepted or rejected, and must simply hope that the underlying PINQAgents accept all of their access requests."* [10](§3.6)

Recall that the key invariant that relates the two runs of the systems on neighbouring data sets (Definition 9) states that the budgets and the scaling factors in the respective environments are equal. This means that they contain *no* information about the sensitive data. This, in turn, means that we can freely permit the program to query them. This would allow the analyst to calculate the cost of queries and to make accuracy decisions relative to the current privacy budget.

Here we briefly outline this extension. We add two new actions to the set of program actions **ProgAct**, namely a query on the sensitivity of a table variable of the form tv ? s, where $s \in \mathbb{N}$, and a query on the global budget, budget ? v where $r \in \mathbb{R}^{\geq 0}$. The transition rules are given in Figure 4.

$$\text{Query sensitivity}\ \frac{P \xrightarrow{tv?s} P'}{\langle P, E, B\rangle \xrightarrow{\tau}_1 \langle P', E, B\rangle}\ \text{where } E(tv) = (T, s)$$

$$\text{Query budget}\ \frac{P \xrightarrow{\mathsf{budget}?B} P'}{\langle P, E, B\rangle \xrightarrow{\tau}_1 \langle P', E, B\rangle}$$

Fig. 4. Budget- and Scaling-Factor-Query

8 Conclusion

We started by presenting some shortcomings(gaps) between the theory of differential privacy and the implementation of PINQ framework. To verify privacy assurance of analysis written in PINQ framework and to address the mentioned concerns, we introduced an idealised model for the implementation of PINQ. In the model, only PINQ's internal implementation has direct access to the sensitive data. An analysis written in this framework has indirect access to the protected system by calling some limited well defined/crafted interface APIs. In addition to the standard PINQ APIs, we extended the model with our own proposed APIs responsible to retrieve scaling factor and the budget from the protected environment. Furthermore we instantiated the definition of differential privacy to prove any analysis constructed in this setting and its communications with protected system would not violate the privacy guarantee promised by PINQ.

We believe that our model (and our general approach to modelling such systems) could be of benefit to formalise emerging variants on the PINQ framework, such as wPINQ [11], or Streaming PINQ [14].

References

[1] Gilles Barthe, Boris Köpf, Federico Olmedo, and Santiago Zanella Béguelin. Probabilistic relational reasoning for differential privacy. *ACM Trans. Program. Lang. Syst*, 35(3):9, 2013.

[2] D. Clark and S. Hunt. Noninterference for deterministic interactive programs. In *Workshop on Formal Aspects in Security and Trust (FAST'08)*, volume 5491 of *LNCS*, 2009.

[3] Cynthia Dwork. Differential privacy. In *ICALP (2)*, volume 4052 of *LNCS*, pages 1–12. Springer, 2006. ISBN 3-540-35907-9.

[4] Cynthia Dwork. A firm foundation for private data analysis. *Commun. ACM*, 54(1), January 2011.

[5] Cynthia Dwork. Differential privacy: A survey of results. In *Theory and Applications of Models of Computation*, pages 1–19. Springer, 2008.

[6] Hamid Ebadi, David Sands, and Gerardo Schneider. Differential privacy: Now it's getting personal. In *Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '15. ACM, 2015.

[7] Marco Gaboardi, Andreas Haeberlen, Justin Hsu, Arjun Narayan, and Benjamin C. Pierce. Linear dependent types for differential privacy. In *Proceedings of the 40th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'13)*, January 2013.

[8] Andreas Haeberlen, Benjamin C Pierce, and Arjun Narayan. Differential privacy under fire. In *USENIX Security Symposium*, 2011.

[9] Frank McSherry. Privacy integrated queries: An extensible platform for privacy-preserving data analysis. *Commun. ACM*, 53(9):89–97, September 2010. ISSN 0001-0782.

[10] Frank D McSherry. Privacy integrated queries: an extensible platform for privacy-preserving data analysis. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*, pages 19–30. ACM, 2009.

[11] Davide Proserpio, Sharon Goldberg, and Frank McSherry. Calibrating data to sensitivity in private data analysis. *40th International Conference on Very Large Data Bases, VLDB'14*, 7(8):637–648, 2014.

[12] Jason Reed and Benjamin C Pierce. Distance makes the types grow stronger: a calculus for differential privacy. *ACM Sigplan Notices*, 45(9):157–168, 2010.

[13] Michael Carl Tschantz, Dilsun Kaynar, and Anupam Datta. Formal verification of differential privacy for interactive systems (extended abstract). *Electron. Notes Theor. Comput. Sci.*, 276:61–79, September 2011.

[14] Lucas Waye. Privacy integrated data stream queries. In *Proceedings of the 5th annual conference on Systems, programming, and applications: software for humanity*. ACM, 2014.

[15] J. Todd Wittbold and Dale M. Johnson. Information flow in nondeterministic systems. In *IEEE Symposium on Security and Privacy*, pages 144–161, 1990.

1 Proof of Theorem

Proof. Assume $\mathbf{Init}(T, B) \stackrel{t}{\Rightarrow}_p \mathbb{C}$. We proceed by induction on the length of the trace t, and by cases according to the last step of the trace.

Base case: $t = []$. In this case $p = q = 1$ and $\mathbb{C} = \mathbf{Init}(T, B)$ and $\mathbb{C}' = \mathbf{Init}(T', B)$. So $\varepsilon = \varepsilon' = 0$ and $\mathbb{C} \sim \mathbb{C}'$.

Inductive step: $t = t_1 a$. Suppose that $\mathbf{Init}(T, B) \stackrel{t_1}{\Rightarrow}_{p_1} \langle P_1, E_1, B_1\rangle \stackrel{a}{\rightarrow}_{p_2} \langle P, E, B\rangle = \mathbb{C}$, and hence that $p = p_1 p_2$.

The induction hypothesis gives us q_1, P_1, E_1' and ε_1 such that

$$\mathbf{Init}(T', B) \stackrel{t_1}{\Rightarrow}_{q_1} \langle P_1, E_1', B_1\rangle \tag{1}$$

$$E_1 \sim E_1' \tag{2}$$

$$p_1 \leq q_1 . \exp(B - \varepsilon_1) \tag{3}$$

by cases that is applied to the rule as the last transition ($\langle P_1, E_1, B_1\rangle \stackrel{a}{\rightarrow}_{p_2} \langle P, E, B\rangle$) we have $p_2 = 1$ except for query execution and that $\langle P_1, E_1', B_1\rangle \stackrel{a}{\rightarrow}_1 \mathbb{C}'$ for some $\mathbb{C}'$. In those cases it follows that $p \leq q \cdot \exp(B - \varepsilon)$ by taking $\varepsilon = \varepsilon_1$ and using (3).

Case 1: Silent. In this case $a = \tau$ and $P_1 \stackrel{\tau}{\rightarrow} P$.

$$\langle P_1, E_1, B_1\rangle \stackrel{\tau}{\rightarrow}_1 \mathbb{C} = \langle P_1, E_1, B_1\rangle$$

$$\langle P_1, E_1', B_1\rangle \stackrel{\tau}{\rightarrow}_1 \mathbb{C}' = \langle P_1, E_1', B_1\rangle$$

It follows directly from (2) that $\mathbb{C} \sim \mathbb{C}'$.

Case 2: Assign. Here $P_1 \xrightarrow{t:=F(t_1,\ldots,t_n)} P$, and so we have

$$\mathbb{C} = \langle P, E_1[tv \mapsto (T, s)], B_1\rangle$$

$$\mathbb{C}' = \langle P, E_1'[tv \mapsto (T', s)], B_1\rangle$$

where for $i \in (1, \ldots, n)$

$$E_1(tv_i) = (T_i, s_i)$$

$$E_1'(tv_i) = (T_i', s_i')$$

$$\mathsf{stability}(F) = (c_1, \ldots, c_n)$$

$$s = \sum_{\sum_i^n} c_i \times s_i$$

$$T = [\![F]\!](T_1, \ldots, T_n)$$

$$T' = [\![F]\!](T_1', \ldots, T_n')$$

From (2) we have $E_1(tv_i) \sim E_1'(tv_i)$ which means $s_i = s_i'$ and $T_i \sim_{s_i} T_i'$. Using similarity definition and Lemma 1 we have $T \sim_s T'$ and hence we have $\mathbb{C} \sim \mathbb{C}'$.

Case 3: Query. The result of query execution depends on the remained budget and the sensitivity of the table that the query is executed on. If privacy budget is insufficient an exception is thrown to inform the program about the shortage of budget, otherwise each query in the list

of queries will be executed on its corresponding partition and the result of execution is returned as a list of values, $\vec{v}$.

Case 3.1: Query(run out of budget). Here we have a rule instance of the form:

$$\text{Query}_{\perp}\ \frac{P \xrightarrow{\mathsf{query}(tv,f,\vec{Q},\varepsilon)\ ?\ \perp} P'}{\langle P,E,B\rangle \xrightarrow{\perp}_1 \langle P',E,B\rangle}\ \text{where}\ \begin{cases} E(tv)=(T,s) \\ \varepsilon\cdot s > B \end{cases}$$

In this case $\mathbb{C} \sim \mathbb{C}'$ and is similar to silent case.

Case 3.2: Query. Similarly we have a rule instance of the form:

$$\text{Query}\ \frac{P \xrightarrow{\mathsf{query}(tv,f,\vec{Q},\varepsilon)\ ?\ \vec{v}} P'}{\langle P,E,B\rangle \xrightarrow{\vec{v}}_p \langle P',E,B-s\cdot\varepsilon\rangle}\ \text{where}\ \begin{cases} E(tv)=(T,s), \quad \varepsilon\cdot s \le B \\ \mathsf{codomain}(f)=\{1,\ldots,n\} \\ \qquad\qquad \vec{v}\in \mathbf{Val}^n \\ T_i=\{s \mid s\in T, f(s)=i\}, \\ \qquad\qquad i\in\{1,\ldots,n\} \\ p=\prod_{i=1}^{n}\Pr[Q_i(\varepsilon,T_i)=v_i] \end{cases}$$

Hence we have a transition : $\langle P_1,E_1,B_1\rangle \xrightarrow{\vec{v}}_{p_2} \mathbb{C}=\langle P,E,B\rangle$ and the analogous transition : $\langle P_1,E_1',B_1\rangle \xrightarrow{\vec{v}}_{q_2} \mathbb{C}'=\langle P,E',B\rangle$. $\varepsilon=\varepsilon_1+(s\cdot\varepsilon_2)$ is the value needed for theorem 1.

For parallel queries on disjoint set we have the following equation:

$$\Pr[P_1 \xrightarrow{\mathsf{query}(tv,f,\vec{Q},\varepsilon)\ ?\ \vec{v}} P] = \prod_{i=1}^{n}\Pr[Q_i(s\cdot\varepsilon_2,T_i)=v_i]$$

Here we need to show that the following inequality is valid:

$$\prod_{i=1}^{n}\Pr[Q_i(s\cdot\varepsilon_2,T_i)=v_i] \le$$

$$\prod_{i=1}^{n}\Pr[Q_i(s\cdot\varepsilon_2,T_i')=v_i'] \times \prod_{i=1}^{n}\exp(\varepsilon_2\times \mid T_i - T_i' \mid)$$

From $\sum_{i=1}^{n}(\mid T_i - T_i' \mid) = s$, we have $\prod_{i=1}^{n}\exp(\varepsilon_2\times \mid T_i - T_i' \mid) \le \exp(\varepsilon_2 \times s)$ which we conclude:

$$\prod_{i=1}^{n}\Pr[Q_i(s\cdot\varepsilon_2,T_i)=v_i] \le \prod_{i=1}^{n}\Pr[Q_i(s\cdot\varepsilon_2,T_i')=v_i'] \times \exp(\varepsilon_2\cdot s)$$

These parallel queries provide $(s\cdot\varepsilon)$-differential privacy which means:

$$p_2 \le q_2\cdot\exp(\varepsilon_2\cdot s)$$

Multiplying two sides of the previous inequality with (3) we get:

$$p_1\cdot p_2 \le q_1\cdot q_2\cdot\exp(\varepsilon_1)\cdot\exp(\varepsilon_2\cdot s)$$

Knowing $B_1 = B-\varepsilon_1$ result in choosing ε to be $\varepsilon = B-\varepsilon_1-(\varepsilon_2\cdot s)$. Finally it is easy to see $\mathbb{C}\sim\mathbb{C}'$ as the proper reduction in the global budget is the only change in the configuration. □

Paper II : Differential Privacy: Now it's Getting Personal

Symposium on Principles of Programming Languages, POPL15

Differential Privacy: Now it's Getting Personal

Hamid Ebadi[1], David Sands[1], and Gerardo Schneider[1,2]

[1] Department of Computer Science and Engineering,
Chalmers University of Technology
Gothenburg, Sweden
{hamide,dave}@chalmers.se
[2] Department of Computer Science and Engineering,
Gothenburg University of Technology
Gothenburg, Sweden
gerardo@cse.gu.se

Abstract. Differential privacy provides a way to get useful information about sensitive data without revealing much about any one individual. It enjoys many nice compositionality properties not shared by other approaches to privacy, including, in particular, robustness against side-knowledge.
Designing differentially private mechanisms from scratch can be a challenging task. One way to make it easier to construct new differential private mechanisms is to design a system which allows more complex mechanisms (programs) to be built from differentially private building blocks in principled way, so that the resulting programs are guaranteed to be differentially private by construction.
This paper is about a new accounting principle for building differentially private programs. It is based on a simple generalisation of classic differential privacy which we call Personalised Differential Privacy (PDP). In PDP each individual has its own personal privacy level. We describe ProPer, an interactive system for implementing PDP which maintains a privacy budget for each individual. When a primitive query is made on data derived from individuals, the provenance of the involved records determines how the privacy budget of an individual is affected: the number of records derived from Alice determines the multiplier for the privacy decrease in Alice's budget. This offers some advantages over previous systems, in particular its fine-grained character allows better utilisation of the privacy budget than mechanisms based purely on the concept of global sensitivity, and it applies naturally to the case of a live database where new individuals are added over time.
We provide a formal model of the ProPer approach, prove that it provides personalised differential privacy, and describe a prototype implementation based on McSherry's PINQ system.

1 Introduction

Differential privacy is a relatively new notion of privacy [5, 7, 6]. The theory shows that by adding the right amount of noise to statistical queries, one can get useful results at the same time as providing a quantifiable notion of privacy. Its definition does not involve a syntactic condition on the data itself, but rather it is a condition formed by comparing the results of a query on any database with or without any one individual: a query Q (a randomised function) is ε-differentially private if the difference in probability of any query outcome on a data-set only varies by a factor of e^{ε} (approximately $1 + \varepsilon$ for small ε) whenever an individual is added or removed.

Research on differential privacy has developed a variety of query mechanisms that provide differential privacy for a useful range of statistical problems. A few works have focussed more on composition principles that allow new differential private mechanisms to design a system which allows more complex mechanisms (programs) to be built from differentially private building blocks in principled way, so that the resulting programs are guaranteed to be differentially private by construction [17, 12, 24]. PINQ [17] is the starting point for the present work.

PINQ-style Global Privacy Budget PINQ is an implementation of interactive differential privacy which ensures, at runtime, that queries adhere to a global privacy budget. Third-party client code freely decides how sensitive data sets should be processed and queried. The run-time system ensures that this does not break a specified privacy budget ε. PINQ builds on a collection of standard differentially private primitive queries, together with simple composition principles – mathematical properties enjoyed by the definition of differential privacy. One central principle is that multiple queries (e.g. with differential privacy ε_1 and ε_2 respectively) have an additive effect ($\varepsilon_1 + \varepsilon_2$) on the overall differential privacy. Another central idea is to track *sensitivity* of functions to measure how much a change in the input might affect the value of the data. Together, these components allow the system to track how much to deduct from the global privacy budget on each invocation of a primitive query.

Limitations of the Global Privacy Budget In a batch system where all computations are described up-front as a monolithic program, a global budget is reasonable. In an interactive system, however, there are several limitations to this style of accounting. Imagine a scenario involving a large data set of individuals – a cross-set of the population – containing various information about health and lifestyle. Let us suppose, further, that we aim for ε-differential privacy for some specified value of ε. On Monday the analyst selects all the people from the database who have a particular blood type, AB-negative, and constructs an algorithm which extracts

information about them as part of medical research. Since just 0.6% of the population have this blood type, the proportion of the database involved in this study is relatively small, but the database is known to be big enough for it to be meaningful. Let us suppose that the cost of this analysis, according to the system, is ε_1. Now on Tuesday the analyst gets a new task, to extract information about the lifestyle of smokers, towards an advertising campaign for nicotine gum. This is a significantly larger portion of the database, possibly overlapping Monday's research group. The analyst has $\varepsilon - \varepsilon_1$ left to spend. If ε_1 is large, the analyst has blown the budget by analysing the small group, even though that study did not touch the data of the larger part of the population. PINQ offers a way around this problem by adding nonstandard database primitives. Here we would partition the data into (AB^-, not AB^-) and perform the two studies in parallel, with cost being the maximum of the cost of the two studies.

This leads to a batch-style reasoning and an unnatural programming style. But it also has another limitation. What if the database is live – we obtain new data over time, or if data is continually being added? A global budget forces us to be unnecessarily pessimistic about new as-yet unexploited data.

Personalised Differential Privacy This paper addresses these issues by offering a simple generalisation of differential privacy called *personalised differential privacy (PDP)* which permits each individual to have a personalised privacy budget. The definition supports generalised versions of the composition principles upon which systems like PINQ are based (§2), and moreover enjoys a number of properties which allow for less wasteful compositional principles (§3). For example, any query about the drinking habits of adults offers 0-differential privacy for Adrian, aged 13, as it does for any records of individuals which enter the database after the query has been made.

From these principles we design a system, in the style of PINQ, called ProPer (**Pro**venance for **Per**sonalised Differential Privacy- §4). The ProPer system maintains a personal budget for every record entering the system. Instead of using sensitivity, ProPer tracks the *provenance* of every record in the system, and uses the exact provenance to calculate how a query should affect the budgets of the individuals. Unlike PINQ, the system is described as an abstract formal model for which we prove personalised differential privacy. This is important because the correctness of ProPer is not obvious for two reasons. Firstly, the individual budgets become highly sensitive and how we handle them is novel. More specifically, if a query involves records that would break the budget of an individual they are silently dropped from the data set upon which the query is calculated. In the example above, Tuesday's analysis of smokers will automatically exclude data derived from any diseased individuals

as soon as the cost of the queries exceeds their budgets. Secondly, it is necessary to restrict the domain of computations over data sets to a class which guarantees that the provenance of any derived record is *affine* (zero or one record), otherwise the number of records which might get excluded due to a small change in the input might be too big to give privacy guarantees.

The approach is suitable for integration with other systems, since we assume the existence of basic primitives providing classical differential privacy. We have implemented a prototype of the ProPer approach which extends the PINQ system (§6) with personalised budgets and the ability to input live data. We compare the performance of our provenance-based implementation with PINQ to show that the runtime overhead is not significant.

We conclude with a discussion of related work (§7), a summary of our contributions (§8), and the current limitations of the approach as well as directions for future work.

2 Differential Privacy

We begin by reviewing the classic definition of differential privacy, and its simple composition principles that allow the construction of new differentially-private algorithms from existing components. In this work the "data sets" will abstract representations of databases, modelled simply as multisets over some unspecified set of records. When we say that A and B differ on at most one record, also written $A \sim B$, we mean that they are either identical, or one can be obtained from the other by adding a single record.

Definition 1. *A randomised function Q provides ε-differential privacy if for all data sets $A \sim B$, and any set of possible outputs $S \subseteq \mathsf{range}(Q)$, we have:*

$$Pr[Q(A) \in S] \leq Pr[Q(B) \in S] \times e^{\varepsilon}$$

Thus the likelihood of a given output to a query Q only changes by a quantifiable amount with or without any individual. The smaller the ε the better the privacy guarantee for the individual. The literature contains many examples of differentially private aggregate operations on data, achieving an appropriate balance between privacy and utility by the principled use of statistical noise. In this work we take the existence of such building blocks as given.

We will adopt the convention of writing Q_ε to denote an ε-differentially private query. Queries will be algorithms rather than just abstract mathematical functions, so we assume that the range of Q is finite and moreover the result of a query forms a discrete probabilistic distribution. Given this,

we can simplify the conditions of the form $Q(A) \in S$ to $Q(A) = v$ for any value in the range of Q.

Definition 1 satisfies a number of useful properties that serve as building blocks for systems enforcing differential privacy [17], which we outline informally here.

Query Composition If we apply two queries Q_{ε_1} and then Q_{ε_2} to a data set, then the combined result is $\varepsilon_1 + \varepsilon_2$ differentially private. This result holds even if Q_{ε_2} is chosen in response to the result of Q_{ε_1}. Although useful, the sequential query composition principle can be very wasteful. If two queries are applied to disjoint sets of data then the privacy loss is the maximum of the privacy losses of the two queries. This observation prompted McSherry to include this as a nonstandard *parallel query* operation in PINQ.

Pre-processing and Sensitivity What if we transform data before applying a query? A key compositionally concept is the *sensitivity* of a function [5] (it is also a key concept in the design of primitive differentially private operations [8], although that is not our focus here). Roughly speaking, a function f has sensitivity c (also known as *stability* c [17]) if whenever the distance between two inputs is n, the distance between the results of applying f is at most $c \times n$. For (multi-)sets, the distance between A and B is just the size of their symmetric difference. Not all functions have a bounded sensitivity. For example consider the cartesian product of two data sets: adding a record to one data set can add unboundedly many elements to the result – it depends on the size of the other argument.

We dub the following property the *sensitivity composition principle*:

> If data-set transformer F has sensitivity c, then $Q_\varepsilon \circ F$ is $(\varepsilon \cdot c)$-differentially private.

The proof of this follows easily from the following scaling property of differential privacy: if data sets A and B differ by k elements, then

$$Pr[Q(A) \in S] \leq Pr[Q(B) \in S] \times e^{(k \cdot \varepsilon)}.$$

Post-processing A very simple – perhaps obvious – property is that a differentially private query is robust under post-processing:

> For any function F, $F \circ Q_\varepsilon$ is ε-differentially private.

This means that post-processing the result of a query cannot extract more private information than the query itself released. This is also the case when F is chosen in response to other queries – or the result of other "side knowledge" about the data set.

3 Personalised Differential Privacy

Now we turn to the main concept introduced in this paper, *Personalised* or "big epsilon" differential privacy, and its analogous compositionality properties.

Definition 2 (Personalised (Big Epsilon) Differential Privacy). *We say that data sets A and B differ on record r, written $A \overset{r}{\sim} B$, if A can be obtained from B by adding the record r, or vice-versa.*

Let $\mathscr{E}$ be a function from records to non-negative real numbers. A randomized query Q provides $\mathscr{E}$-differential privacy *if for all records r, and all $A \overset{r}{\sim} B$, and any set of outputs $S \subseteq \mathsf{range}(Q)$, we have: $Pr[Q(A) \in S] \leq Pr[Q(B) \in S] \times e^{\mathscr{E}(r)}$*

Personalised differential privacy allows each individual (record) to have its own personal privacy level. This may turn out to be a useful concept in its own right, but its main purpose in this work is as a generalisation that permits a more fine-grained accounting in the construction of classical differentially private mechanisms, and one which plays well with dynamic databases. The following proposition summarises the relation to "small-epsilon" differential privacy:

Proposition 1.

(i) If Q is ε-differentially private, then Q is $\lambda x.\varepsilon$-differentially private.
(ii) If Q is $\mathscr{E}$-differentially private, and $\sup(\mathsf{range}(\mathscr{E})) = \varepsilon$ then Q is ε-differentially private.

Now we consider the composition principles analogous to those above. We keep the presentation informal since we will not apply these principles directly in our formal developments – rather they provide an intuition behind the approach. Most of the principles above generalise to personalised differential privacy.

Query Composition In the sequential composition of queries, if Q_1 and Q_2 are $\mathscr{E}_1$ and $\mathscr{E}_2$-differentially private, respectively, then applied in sequence they yield a $\lambda x.\mathscr{E}_1(x) + \mathscr{E}_2(x)$-differentially private query. For parallel queries let us be a little more precise:

Let $\{R_i\}_{i \in I}$ be a partition of the set of all records, and $\{Q_i\}_{i \in I}$ be a set of queries. we define a parallel query $P(A) = \Pi_{i \in I} Q_i(A \cap R_i)$ where Π is just the n-ary cartesian product of sets. Now we have the following natural generalisation of the parallel query principle:

> If Q_i is $\mathscr{E}_i$-differentially private then P is $\mathscr{E}$-differentially private, where $\mathscr{E}(r) = \mathscr{E}_i(r)$ if $r \in R_i$.

Now we introduce the first specialised principle which takes advantage of the fine-grained nature of personalised differential privacy, the *selection principle*:

> For set A, define $\mathsf{select}_A(x) = x \cap A$. If Q is $\mathscr{E}$-differentially private, then $Q \circ \mathsf{select}_A$ is $\mathscr{E}[r \mapsto 0 \mid r \notin A]$-differentially private.

Here $\mathscr{E}[r \mapsto 0 \mid r \notin A]$ denotes the function which maps every element outside A to 0, and behaves as $\mathscr{E}$ otherwise. In simple terms, a query which operates on A is perfectly private for individuals outside of A. In contrast, the composition principle of ε-differential privacy has nothing helpful to say here: the sensitivity of the selection function is 1.

How does this help us? It can show how the sequential composition principle for $\mathscr{E}$-differential privacy gives greater accounting precision. Specifically, parallel composition is simply no longer necessary to give a reasonably accurate estimate of privacy cost. Suppose we compute $P(A)$ by sequentially computing $Q_i \circ \mathsf{select}_{R_i}$. Then the sequential composition principle calculates the cost of this iterated sequential composition as

$$\lambda x.\Sigma_{i \in I}(\text{if } x \in R_i \text{ then } \mathscr{E}_i(x) \text{ else } 0)$$

which is precisely the cost calculated for the parallel query.

Sensitivity Composition The sensitivity composition principle also lifts into the world of personalised differential privacy:

> If data-set transformer F has sensitivity c, and Q is $\mathscr{E}$-differentially private, then $Q \circ F$ is $\lambda x.(\mathscr{E}(x) \times c)$-differentially private.

The proof analogously follows easily from the following scaling property: If data sets A and B differ on elements C, then

$$Pr[Q(A) \in S] \leq Pr[Q(B) \in S] \times \exp(\Sigma_{r \in C}\mathscr{E}(r)).$$

The Personalised Sensitivity Principle A key feature of personalised differential privacy is that it supports a fine-grained composition property for a large and important class of functions, namely functions that are union preserving. A function F from multisets to multisets is union preserving if $F(A \uplus B) = F(A) \uplus F(B)$, $A \uplus B$ denotes the additive union of multisets A and B. In standard relational algebra, for example, all functions are union preserving in each of their arguments, with the exception of the (multi-)set difference operator which is not union preserving in its second argument. Complex union-preserving functions may be built from simple ones as they are closed under compositions. You will read more about supported compositions in section 4.3.

The characteristic property of union preserving functions is that their behaviour can be completely characterised by their behaviour on individual records. This gives us a completely precise way to compute the influence of a single record on the result of the function, since $F(A \uplus \{r\}) = F(A) \uplus F\{r\}$. This leads us to the following.

Lemma 1. *If F is a union-preserving function and Q is $\mathscr{E}$-differentially private, then $Q \circ F$ is $(\lambda x.\Sigma_{s \in F\{x\}} \mathscr{E}(s))$-differentially private.*

Proof. Follows easily from the scaling property.

Taking $\mathscr{E} = \lambda r.\varepsilon$ yields the following useful corollary which is the core of our approach to combining existing differentially private mechanisms with our personalised approach:

Corollary 1. *If F is a union-preserving function and Q is ε-differentially private, then $Q \circ F$ is $(\lambda x.\mathsf{size}(F\{x\}) \times \varepsilon)$-differentially private.*

4 ProPer: Provenance for Personalised Privacy

In this section we provide an abstract model of the ProPer system. The ProPer system encapsulates sensitive databases and manages computations over them via an imperative API. It guarantees (as we shall prove) $\mathscr{E}$-differential privacy for the records which enter the system.

Descriptive vs Prescriptive Systems The principles described in the previous section are a guide as to how we can build a system that allows non-expert analysts to compose new differentially private algorithms from differentially private components. In principle such systems can be of two kinds: *descriptive*, or *prescriptive*. In a descriptive system we can apply the principles to compute and *report* on the level of privacy achieved by a given run of a program. A prescriptive system, on the other hand, is given a goal – an amount of privacy that is to be achieved, and the system must use the principles and ensure that the privacy stays within the bound.[3] In a dynamic system like PINQ, however, it is more natural to describe a prescriptive system – one which does not violate a preconceived level of privacy.

In a prescriptive system the desired amount of privacy can be thought of as a *budget*, and in the literature it is often referred to as such. For ProPer this is an amount of privacy per record as described by some function $\mathscr{E}$. But the principles described above know nothing of budgets – they are purely descriptive. It is therefore important to design a mechanism which is private even when the program fails to meet the intended goals. With personalised differential privacy this is a crucial question – because the budget itself is clearly a sensitive object. In a nutshell, the ProPer approach involves tracking the provenance of each record in any intermediate table (on which sensitive input record does it depend), and by silently dropping records from the arguments to statistical queries if

[3] In an approach based on static analysis, such as the type-system of Fuzz [12], one could say that these two approaches are unified.

the presence of those records would break the privacy budget of some individual. The provenance information is used to make that link.

We begin with an informal overview before describing the system in formal terms.

4.1 Overview of ProPer

The ProPer system is described in terms of two main components: the *protected system* which stores all sensitive data and its derivatives, and mediates all computation over that data, and *the client program* which queries the sensitive data, requests computations to be performed over the sensitive data, and drives the inclusion of new input records into the protected system. An illustration of the architecture is given in Figure 1.

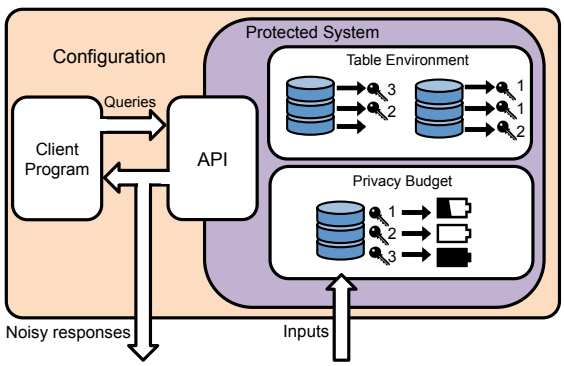

Fig. 1. ProPer System Structure

From the viewpoint of a client program, the system just stores tables. Tables are referenced via *table variables*. A client program will issue a command to the protected system phrased in terms of table variables. These commands will represent *transformations* such as "Select all females from table A and assign the result to table B", or "Input some new records and assign them to table C", or *primitive differential-private queries* such as "Return the number of records in table C, with 0.5-differential privacy.".

The records that are input are the subject of our privacy concerns. We refer to those records as *individuals*. To provide $\mathscr{E}$-differential privacy for the individuals, the protected system needs to maintain more information than just the mapping between table variables and tables. For each individual r that has been input to the system so far, a *privacy budget*

for r needs to be maintained. Initially the budget will be $\mathscr{E}(r)$. As queries are performed over time the budget for each individual may decrease.

There are two key issues that the system must address: (i) how much should the budget for each individual be decreased when a table is queried, and (ii) how do we prevent the budget from becoming negative (which would imply that we have violated privacy).

The solution to this is to track, together with each record, its *provenance*. The provenance of a given record is just the individual from which that record was derived (if any). Problem (i) is then solved by noting that the cost for individual r is the privacy cost of the query multiplied by the number of elements in the table which have provenance r (c.f. Corollary 1).

Before we can provide the formal definitions of the above sketch we need to define our basic domains and introduce some suitable notation.

4.2 Preliminary Definitions and Notation

Given sets A and B, $A \rightharpoonup B$ denotes the set of partial functions between A and B with finite domain. If f is (partial) function then $f[x \mapsto y]$ denotes the (partial) function which maps x to y and behaves as f for all other arguments. We will also write partial functions (and updates) using a set-comprehension style notation, e.g. $[x \mapsto x+1 | x \in \{-1, 0, 1\}]$.

We assume an untyped set of records Rec, ranged over by r, s, etc. We will work extensively with multisets, in particular multisets of records. For a set A, we write $\mathsf{mset}(A)$ to denote the set of multisets over A (isomorphic to functions $A \to \mathbb{N}$). More specifically, the set of tables Table is defined to be $\mathsf{mset}(\mathrm{Rec})$.

It will be convenient to introduce some notation for working with multisets. We use multiset brackets such as e.g. $A = \{\!|5, 5, 6|\!\}$, denoting a multiset A containing two copies of 5 and one 6. We write multiset membership $a \overset{n}{\in} A$ to mean that there are exactly n copies of a in A, and $a \in A$ means $\exists n > 0.\, a \overset{n}{\in} A$. Analogous to set-comprehensions (set-builder notation) we will use multiset comprehensions, with the ability to express multiplicities. For example:

$$\{\!|x^{[n]} \mid x \overset{n}{\in} \{\!|5, 5, -5|\!\} \wedge x > 0 \,|\!\}\} = \{\!|5, 5|\!\}.$$

But note that multiplicities may "sum up" as in this example:

$$\{\!|0 \times x^{[n]} \mid x \overset{n}{\in} \{\!|5, 5, -5|\!\} \,|\!\}\} = \{\!|0, 0, 0|\!\}.$$

Given multisets A and B, we write $A \uplus B$ to denote the additive union, which is the least multiset such that whenever $a \overset{n}{\in} A$ and $a \overset{m}{\in} B$ then $a \overset{m+n}{\in} (A \uplus B)$.

Multi-Relations Binary relations are to sets as multi-relations are to multisets. In other words a multi-relation is just a notation for a multiset of pairs [13].

We will use the concept of multi-relation extensively to model the records of a table together with their provenance. For example, a table containing just three copies of a record r, two of which where derived from individual Alice and one from individual Bob, will be modelled by a multi-relation $\{\!|(r, \text{Alice}), (r, \text{Alice}), (r, \text{Bob})|\!\}$.

Before we show how we use this in practice we need some notation to make reasoning with multi-relations more palatable.

Formally, given sets A and B we write $A \leftrightarrow B$ to denote $\mathsf{mset}(A \times B)$. If $R \in A \leftrightarrow B$ then we write $a \overset{n}{R} b$ to mean a is related to b, n times, i.e., $(a, b) \overset{n}{\in} R$.

Definition 3 (Operations on Multi-relations). *Let R and S range over $X \leftrightarrow X$ for some X, let be T is a subset of X, and $U \in \mathsf{mset}(X)$. We define the following operations involving multi-relations:*

Domain of a Relation $\mathsf{dom}(R) \stackrel{\text{def}}{=} \{\!|a^{[n]} \mid (a, b) \overset{n}{\in} R|\!\}$

Relation Composition $R \,;\, S \stackrel{\text{def}}{=} \{\!|(a, c)^{[m \cdot n]} \mid a \overset{n}{R} b \wedge b \overset{m}{S} c|\!\}$

Application $R \bullet U \stackrel{\text{def}}{=} \{\!|a^{[n \cdot m]} \mid a \overset{n}{R} b \wedge b \overset{m}{\in} U|\!\}$

Right Restriction $R \rhd T \stackrel{\text{def}}{=} \{\!|(a, b)^{[n]} \mid a \overset{n}{R} b \wedge b \in T|\!\}$

Example 1. Given the following two multi-relations:

$$R = \{\!|(x, y), (x, w), (x, w)|\!\} \qquad S = \{\!|(y, z), (w, z), (w, v)|\!\}$$

then $\mathsf{dom}(S) = \{\!|y, w, w|\!\}$, $R \,;\, S = \{\!|(x, z)^{[3]}, (x, v)^{[2]}|\!\}$, $R \rhd \{w\} = \{\!|(x, w)^{[2]}|\!\}$, and $R \bullet \{\!|w^{[2]}, y|\!\} = \{\!|x^{[5]}|\!\}$.

The following properties are easily verified.

Proposition 2. *For all multi-relations R, R', S, and S', and set T,*
(i) Composition is $\uplus$-preserving in both arguments:

$$(R \uplus R') \,;\, S = (R \,;\, S) \uplus (R' \,;\, S)$$
$$R \,;\, (S \uplus S') = (R \,;\, S) \uplus (R \,;\, S'),$$

(ii) restriction is $\uplus$-preserving in its first argument:

$$(R \uplus S) \rhd T = (R \rhd T) \uplus (S \rhd T),$$

(iii) restriction and composition associate as follows:

$$(R \,;\, S) \rhd T = R \,;\, (S \rhd T),$$

(iv) and finally:

$$\mathsf{dom}(R \rhd T) = R \bullet T$$

Note that, as in (iv), we will freely use sets as if they were multisets without making the obvious injection operation explicit.

4.3 Provenance Tracing

As mentioned, we will track the provenance of each record derived. The key idea to achieve personalised differential privacy is that the provenance of a given record must be at most one record. We call this *affine* provenance.

Supported operations As we explained, in this setting, records' provenance should be affine. This is achieved by simply requiring that all transformations are unary and $\uplus$-preserving i.e., transformations F for which $F(A \uplus B) = F(A) \uplus F(B)$. This guarantees that provenance can be tracked by observing the action of F on singletons, and (ii) provenance will always be a single element.

To give a simple syntactic characterisation of a class of unary $\uplus$-preserving functions, we can use a grammar of terms built from the standard operations of relational algebra, used here over multisets. The basic operators of relational algebra, transposed to multisets, are the set operations (multiset union $\uplus$, set difference $-$, cartesian product $\times$), together with record selection σ_p, which selects all elements satisfying property p, and projection π_a which transforms each row by retaining only the columns given by schema a. We omit the details of the definitions and refer to [11] for definitions of multiset variants of these standard operations.

Definition 4 (Affine Relational Terms). *Let V range over sets of variables and T over literal multisets. We define a family of variable-indexed relational algebra terms A_V by the following grammar:*

$$\begin{aligned} A_V ::= {} & x \ (\in V) \mid A_V \uplus A_V | A_V - A_{\{\}} | A_V \times A_{\{\}} | A_{\{\}} \times A_V \\ & \mid \sigma_p(A_V) \mid \pi_a(A_V) \mid T \end{aligned}$$

Theorem 1. *Any multiset transformation F defined by $F(x) = A_{\{x\}}$ is $\uplus$-preserving.*

The proof follows by induction on the definition of $A_{\{x\}}$ using the union-preserving properties of the operators. The restrictions imposed by the grammar are due to the facts that all the operations preserve unions in each argument individually, except for the second argument of set difference, and that $\uplus$ preserves unions across its arguments simultaneously, whereas $\times$ does not (i.e. $(A \uplus A') \times (B \uplus B') \neq (A \times B) \uplus (A' \times B')$).

Provenance Tables The fact that we will track affine provenance leads us to define a *provenance table* as a table in which the affine provenance of each element is recorded.

Definition 5 (Provenance Table). *A provenance table is a multi relation of type* $\text{ProvTable} \stackrel{\text{def}}{=} \text{Rec} \leftrightarrow \text{Rec}_\perp$*, where* $\text{Rec}_\perp \stackrel{\text{def}}{=} \text{Rec} \cup \{\perp\}$ *for some distinguished non-record $\perp$.*

The underlying table T represented by a provenance table D is obtained by simply taking the domain of D, i.e. $T = \mathsf{dom}(D)$. The provenance of each element r of the table T is given by the element to which they are related, viz, if rDs then there is a copy of r in T that has provenance s. If some record r is related to $\perp$ this signifies that r is present in the table, but that it has no provenance (i.e. it was not derived from any individual).

In the remainder of this section we introduce the notation and techniques necessary to permit provenance to be traced across computation.

How do we build and maintain provenance tables? We need a way to create provenance tables from new tables, and we need a way to construct the provenance table of a table produced by a transformation applied to a (provenance) table.

When new records of individuals enter the system then their provenance table has a simple form: the provenance of each record is itself. When we create a new literal table (i.e. where the elements do not depend on individuals) then each record has provenance $\perp$. The following notation for these cases will be useful:

Definition 6. *For a set of records R, define the* identity provenance table

$$\mathsf{Id}_R \stackrel{\text{def}}{=} \{\!|(r, r) \mid r \in R|\!\}.$$

For a table T define the constant provenance table

$$\mathsf{Const}_T \stackrel{\text{def}}{=} \{\!|(r, \perp)^{[n]} \mid r \overset{n}{\in} T|\!\}.$$

The final building block is to show how to lift a function F which computes over tables to a function $\hat{F}$ which computes over provenance tables so that (in particular) the following diagram commutes:

$$\begin{array}{ccc} D & \xrightarrow{\hat{F}} & D' \\ {\scriptstyle \mathsf{dom}}\,\vdots & & \downarrow\,{\scriptstyle \mathsf{dom}} \\ T & \xrightarrow[F]{} & T' \end{array}$$

Definition 7. *Given a $\uplus$-preserving function $F \in$ Table $\to$ Table, define $\hat{F} \in$ ProvTable $\to$ ProvTable by*

$$\hat{F}(D) \stackrel{\text{def}}{=} \widetilde{F}\,;\, D$$

where $\widetilde{F} \in \mathrm{Rec} \leftrightarrow \mathrm{Rec}$ is defined by $t \overset{n}{\widetilde{F}}\, s \Leftrightarrow t \overset{n}{\in} F(\{s\})$

The fact that the diagram above commutes is captured as follows:

Lemma 2 (functional correctness). $\mathsf{dom}(\hat{F}(D)) = F(\mathsf{dom}(D))$

Proof. First we show that the relational representation of F and the relational application operator behave as expected:

$$\widetilde{F} \bullet A = F(A) \tag{1}$$

$$\begin{aligned}
\widetilde{F} \bullet A &= \{\!|a^{[m \cdot n]} \mid a \overset{n}{\widetilde{F}} b, b \overset{m}{\in} A |\!\} \\
&= \{\!|a^{[m \cdot n]} \mid a \overset{n}{\in} F(\{b\}), b \overset{m}{\in} A |\!\} && \text{(Def. 7)} \\
&= \{\!|a^{[k]} \mid a \overset{k}{\in} F(A) |\!\} && (F \text{ preserves } \uplus) \\
&= F(A)
\end{aligned}$$

Now we show:

$$\mathsf{dom}(D \,;\, D') = D \bullet \mathsf{dom}(D') \tag{2}$$

$$\begin{aligned}
\mathsf{dom}(D \,;\, D') &= \mathsf{dom}(\{\!|(a,c)^{[m \cdot n]} \mid a \overset{m}{D} b, b \overset{n}{D'} c |\!\}) \\
&= \{\!|a^{[m \cdot n]} \mid a \overset{m}{D} b, b \overset{n}{D'} c |\!\} \\
&= \left\{\!\!\left| a^{[m \cdot k]} \;\middle|\; (a,b) \overset{m}{\in} D \wedge b \overset{k}{\in} \mathsf{dom}(D') \right|\!\!\right\} \\
&= D \bullet \mathsf{dom}(D')
\end{aligned}$$

Finally we calculate:

$$\begin{aligned}
\mathsf{dom}(\hat{F}(D)) &= \mathsf{dom}(\widetilde{F} \,;\, D) \\
&= \widetilde{F} \bullet \mathsf{dom}(D) && \text{(Eq. 2)} \\
&= F(\mathsf{dom}(D)) && \text{(Eq. 1)} \qquad \square
\end{aligned}$$

4.4 The System Model

The semantics of the overall system will be given by a probabilistic transition system described by combining the *client program* with the *protected system*.

Protected System The protected system is a collection of states which encodes four pieces of information:

1. the set of individuals which have been input to the system so far,
2. a *privacy budget* (a positive real) for each of these individuals indicating how much of the personalised privacy remains,
3. a set of *table variables* TVar used to identify intermediate tables computed, and

4. a *table environment* which maps each *table variable* to the provenance table it represents.

The first two items are modelled by a partial function from individuals to budgets, and we assume that the set of table variables is fixed (and at least countable). This leads us to the formal definition of the states:

Definition 8 (Protected System States).

$$\text{States} \stackrel{\text{def}}{=} (\text{TVar} \to \text{ProvTable}) \times (\text{Rec} \rightharpoonup \mathbb{R}^+)$$

Client Program Model We work with an abstract notion of a client program. A program is just a labelled transition system, subject to some restrictions, where the labels – the *actions* – represent the imperative API through which the program interacts with the protected system.

The program model, inspired by PINQ [17], is an imperative program that computes with tables by requesting that the commands $a \in \text{ProgAct}$ are performed. ProgAct is specified in figure 2.

ProgAct ::=	τ	Silent step
	$\mid tv := \text{Expr}$	Assignment
	$\mid Q_\varepsilon(tv)?v$	Primitive Query returning $v \in \text{Val}$

Expr ::=	tv	Table variable
	$\mid F(tv_1 \uplus \cdots \uplus tv_k)$	Transform
	$\mid T$	Table literal, $T \in \text{Table}$
	$\mid \mathsf{input}$	Reference to the input stream

Fig. 2. ProgAct: the labels of the transition system

Here we assume that F ranges over an unspecified set denoting $\uplus$-preserving functions in Table $\to$ Table, and Q_ε ranges over an unspecified set of queries with the convention that Q_ε denotes an ε-differentially private query. Note that we will not formally distinguish the name of a function (as used here to define the set of actions) from its denotation (as used in the specification of the semantics of the system below).

The idea is that programs have no direct access to tables, but make requests for the system to manipulate them on their behalf. This includes making a request for the system to collect new individuals via an input action (and place them in a table variable). The primitive query action is special. Firstly, it does not model a request, but rather a request and its

result all in one. The reason for this is that it allows us to model value passing without needing to introduce any specific syntax for programs. Secondly, the value returned by the query is known to the program, and the program can act on it accordingly. From the perspective of the program and the protected system together, this value will be considered an observable output of the whole system. We also model internal actions of the program (and hence the passage of time) via the traditional silent action τ. A program, then, is just a ProgAct-labelled transition system. However we impose some mild restrictions on the transition system which model the fact that (i) the query operation really is an input operation, so if the program issues a query, there must be a transition for that query with every possible value, and that (ii) the program is fully deterministic, so that at most one type of action is possible in any given state, and the action determines the next state.

Definition 9 (Client Program). *A client program (a program for short) is a labelled transition system* $\langle \mathbb{P}, \rightarrow, P_0 \rangle$ *where* $\mathbb{P}$ *is the set of program states with initial state* P_0 *and transition relation* $\rightarrow \subseteq (\mathbb{P} \times \mathrm{ProgAct} \times \mathbb{P})$*, satisfying the following properties. Firstly it is deadlock free – a program can always make a transition, and secondly it satisfies the determinacy property:*

For all states P*, if* $P \xrightarrow{a} P'$ *and* $P \xrightarrow{b} P''$ *then*

1. *if* $a = b$ *then* $P' = P''$*,*
2. *if* a *is not a query then* $a = b$*,*
3. *if* $a = Q_\varepsilon(tv)?r$ *then* $b = Q_\varepsilon(tv)?r'$ *for some* r'*, and for all* s *there exists a* P_s *such that* $P \xrightarrow{Q_\varepsilon(tv)?s} P_s$*.*

Remark: Implicit parameters To avoid excessive parameterisation of definitions, in what follows we will fix some arbitrary client program $\langle \mathbb{P}, \rightarrow, P_0 \rangle$ and some arbitrary personal budget $\mathscr{E}$ and make definitions relative to these.

Configuration Semantics Now we can provide the semantics of the combination of a program and the protected system – what we will call a *configuration*:

$$\mathbb{C} \in \mathrm{Config} \stackrel{\mathrm{def}}{=} \mathbb{P} \times (\mathrm{TVar} \rightarrow \mathrm{ProvTable}) \times (\mathrm{Rec} \rightharpoonup \mathbb{R}^+)$$

We will write a configuration as a triple $\langle P, E, B \rangle$ where metavariable E ranges over the table environment and B ranges over the budget.

The behaviour of a configuration will be a form of probabilistic labelled transition system whose labels are the values of queries made by the program, the silent transition τ, and the tables of unique individuals which are input by the environment.

$$\text{Input}\ \frac{P \xrightarrow{tv:=\mathsf{input}} P' \qquad T \in 2^{\mathrm{Rec}} \qquad T \cap \mathsf{dom}(B) = \emptyset}{\langle P, E, B\rangle \xrightarrow{T}_1 \langle P', E[tv \mapsto \mathsf{Id}(T)], B[r \mapsto \mathscr{E}(r) \mid r \in T]\rangle}$$

$$\text{Assign}\ \frac{P \xrightarrow{tv:=e} P'}{\langle P, E, B\rangle \xrightarrow{\tau}_1 \langle P', E[tv \mapsto D], B\rangle} \quad \text{where}$$

$$D = \begin{cases} \hat{F}(E(tv_1) \uplus \cdots \uplus E(tv_n)) & \text{if } e = F(tv_1 \uplus \cdots \uplus tv_n) \\ \mathsf{Const}(T) & \text{if } e = T \\ E(tv') & \text{if } e = tv' \end{cases}$$

$$\text{Silent}\ \frac{P \xrightarrow{\tau} P'}{\langle P, E, B\rangle \xrightarrow{\tau}_1 \langle P', E, B\rangle}$$

$$\text{Query}\ \frac{P \xrightarrow{Q_\varepsilon(tv)?n} P'}{\langle P, E, B\rangle \xrightarrow{n}_p \langle P', E, B[r \mapsto B(r) - C(r) \mid r \in A]\rangle} \quad \text{where}$$

$$\begin{aligned} C &= [s \mapsto \varepsilon.\mathsf{size}(E(tv) \bullet \{s\}) \mid s \in \mathsf{dom}(B)] \\ A &= \{r \mid B(r) \geq C(r)\} \\ L &= E(tv) \bullet (A \cup \{\bot\}) \\ p &= \Pr(Q_\varepsilon(L) = n) \end{aligned}$$

Fig. 3. Operational Semantics

Definition 10 (Initial configuration). *The initial configuration is formed from the initial program state, the environment that maps every variable to the empty provenance table, and the empty budget table:*

$$\mathbb{C}_0 \stackrel{\text{def}}{=} \langle P_0, [tv \mapsto \{\!\{\}\!\} \mid tv \in \mathrm{TVar}], []\rangle$$

Definition 11 (Operational Semantics). *The operational semantics of configurations is given by a "probabilistic" labelled transition relation with transitions of the form* $\mathbb{C} \xrightarrow{a}_p \mathbb{C}'$ *where* $a \in \mathrm{Act} \stackrel{\text{def}}{=} \{\tau\} \cup \mathrm{Val} \cup 2^{\mathrm{Rec}}$*, and (probability)* $p \in [0, 1]$*. The definition is given by cases in Figure 3.*

We put "probability" in parentheses because the relation is not *a priori* probabilistic but something that must be proven; this is established in Lemma 3. First we provide some explanation of each nontrivial rule in turn.

[Input] The program requests an input to be made and assigned to a table variable. The rule imposes a constraint on the records T which are input: it must be a *set* of records, and this set must be disjoint from the

records previously input (the domain of B). This reflects the idea that the input records are the subject of privacy and represent unique individuals. The transition of the configuration is labelled with T to record that the environment chose to input T. The probability of the transition is 1, meaning that the choice of input is treated nondeterministically. The configuration is updated in two ways. Firstly, the table is converted to a provenance table by recording that the provenance of each record is itself. Secondly the budget for each new record is initialised from $\mathscr{E}$.

[Assign] The program requests a transformation of existing data. Here we apply the mechanisms developed in Section 4.3 to lift the function (respectively, table) into the world of provenance tables.

[Query] Here the program is requesting the value of a query $Q_\varepsilon(tv)$. To answer the query we must determine the *eligible* records, L, from the table tv, which can safely be involved in this query. To do this we first determine a *Cost map* C, which describes the privacy cost which would be inflicted upon individual r by releasing the query $Q_\varepsilon(tv)$; the cost of an ε-differentially private query on tv to an individual r is ε multiplied by the number of records in tv which have provenance r. Given the cost map, we can determine A, the set of individuals for which this cost is *Acceptable* – i.e. those who have sufficient budget. Finally we can use A to determine L: it is the sub-table of records which depend at most on records in A.

The probability of the transition is inherited from the probability that the query returns that particular value.

4.5 Trace semantics

The transition system on configuration $\mathbb{C} \xrightarrow{a}_p \mathbb{C}'$ is fully probabilistic in that $\mathbb{C}$ and the value of a uniquely determine p and, when $p > 0$, $\mathbb{C}'$. This makes it very straightforward to lift the single-step semantics to a probabilistic trace semantics.

In what follows let σ range over *traces*, sequences of zero or more actions Act^*. The empty trace is denoted by $[]$ and $a\sigma$ denotes the trace starting with a and continuing with σ.

Definition 12 (Trace semantics). *Define the trace transition relation*

$$\Rightarrow \subseteq \mathrm{Config} \times \mathrm{Act}^* \times [0,1] \times \mathrm{Config}$$

inductively by the following rules:

$$\mathbb{C} \overset{[]}{\Rightarrow}_1 \mathbb{C} \qquad \frac{\mathbb{C} \xrightarrow{a}_p \mathbb{C}' \qquad \mathbb{C}' \overset{\sigma}{\Rightarrow}_q \mathbb{C}''}{\mathbb{C} \overset{a\sigma}{\Longrightarrow}_{p.q} \mathbb{C}''}$$

We write $\mathbb{C} \stackrel{\sigma}{\Rightarrow}_p$ to mean $\mathbb{C} \stackrel{\sigma}{\Rightarrow}_p \mathbb{C}'$ for some $\mathbb{C}'$.

Although we have a trace-labelled transition system involving numbers derived from probabilities, it remains to show in what sense we have specified a probabilistic system. We begin with a definition which describes when an input sequence is compatible with a given trace.

Definition 13 (Input Compatibility). *An* input trace i *is a sequence of mutually disjoint sets of records. We say that a trace* t *is* compatible *with* i*, written* $t \vdash i$ *iff the subsequence of inputs in* t *is a prefix of* i*.*

Now we can state the sense in which the transition system is probabilistic: it can be viewed as a probabilistic function of the input and the length of the trace observed:

Lemma 3 (Traces are Probabilistic). *For all input traces* i*, and all* $n \geq 0$*,*

$$\Sigma\{\!|p \mid \mathbb{C}_0 \stackrel{t}{\Rightarrow}_p \ \wedge t \vdash i \wedge \mathsf{size}(t) = n \}\!| = 1$$

I.e. in response to a given input, the possible traces of a program of a given length form a probability distribution.

Proof. (sketch) A key here is the following determinacy property: whenever $\mathbb{C} \stackrel{a}{\rightarrow}_p \mathbb{C}'$ and $\mathbb{C} \stackrel{a}{\rightarrow}_q \mathbb{C}''$ for $p, q > 0$, then $\mathbb{C}' = \mathbb{C}''$ and $p = q$. This can be established by cases according to the transition, and depends crucially on the assumption that the program transitions are deterministic. From this it is straightforward to show that $\mathbb{C} \stackrel{t}{\Rightarrow}_p \mathbb{C}'$ and $\mathbb{C} \stackrel{t}{\rightarrow}_q \mathbb{C}''$ for $p, q > 0$, imply $p = q$ and $\mathbb{C}' = \mathbb{C}''$. The first clause($p = q$) is easily established by cases according to the rule inducing the transition, making use of determinism assumption about programs; the second clause ($\mathbb{C}' = \mathbb{C}''$) then follows easily from the first. Armed with these two properties, the proof follows by induction on n.

So whenever $\mathbb{C} \stackrel{\sigma}{\Rightarrow}_p$, p is the probability of observing σ among traces of the same length and for which the input sequence is the same. We thus write $Pr(\mathbb{C} \stackrel{\sigma}{\Rightarrow}) = p$ to mean $\mathbb{C} \stackrel{\sigma}{\Rightarrow}_p$.

5 ProPer Provides $\mathscr{E}$-Differential Privacy

In this section we establish the main theorem for the system, which states that the trace semantics is an $\mathscr{E}$-differentially private function of its input. In order to state this in a convenient way we introduce some notation.

Definition 14 (r-difference). *For any record* r *and any tables* T *and* T'*, we write* $T \stackrel{r}{\sqsubset} T'$ *to mean that* $T \uplus \{\!|r|\!\} = T'$*. We lift this relation to traces, writing* $\sigma \stackrel{r}{\sqsubset} \sigma'$ *to mean* $\sigma = \sigma_1 T \sigma_2$ *and* $\sigma' = \sigma_1 T' \sigma_2$*, for some* σ_1*,* σ_2*,* T*,* T' *such that* $T \stackrel{r}{\sqsubset} T'$*.*

We will further lift $T \overset{r}{\sqsubset} T'$ to various structures containing tables. In all cases we define the overloaded relation $\overset{r}{\sim}$ to be the symmetric closure of $\overset{r}{\sqsubset}$, so $T \overset{r}{\sim} T'$ iff either $T \overset{r}{\sqsubset} T'$ or $T' \overset{r}{\sqsubset} T$.

So when $\sigma \overset{r}{\sim} \sigma'$ then σ and σ' differ in exactly one element, an input set, and and their difference is exactly the record r.

Theorem 2 ($\mathscr{E}$-differential privacy for all traces). *If $\sigma \overset{r}{\sim} \sigma'$ and $\Pr(\mathbb{C} \overset{\sigma}{\Rightarrow}) = p$ then $\Pr(\mathbb{C} \overset{\sigma'}{\Rightarrow}) = q$ for some q such that $p \leq q \cdot \exp(\mathscr{E}(r))$.*

To prove the theorem (inevitably by an induction over the trace length) we must establish an invariant relation over configurations which generalises $T \overset{r}{\sqsubset} T'$. The most tricky and important of these is the relation between provenance tables.

Definition 15. $D \overset{r}{\sqsubset} D' \Leftrightarrow D \uplus (D' \rhd \{r\}) = D'$

This says that when $D \overset{r}{\sqsubset} D'$, the table represented by D' has all the elements of D, with the same provenance, and that the elements of D' which are not in D all have provenance r.

This relation will be used to express a key invariant in the correctness proof, and can also be thought of (in the main proof) as establishing the correctness of the provenance information. Now we establish some basic properties of this relation on provenance tables.

Proposition 3. $D \overset{r}{\sqsubset} D' \Rightarrow \mathsf{dom}(D') = D' \bullet \{r\} \uplus \mathsf{dom}(D)$

Proof.

$$\begin{aligned} D \overset{r}{\sqsubset} D' &\Rightarrow (D' \rhd \{r\}) \uplus D = D' \\ &\Rightarrow \mathsf{dom}((D' \rhd \{r\}) \uplus D) = \mathsf{dom}(D') \\ &\Rightarrow \mathsf{dom}(D' \rhd \{r\}) \uplus \mathsf{dom}(D) = \mathsf{dom}(D') \\ &\Leftrightarrow D' \bullet \{r\} \uplus \mathsf{dom}(D) = \mathsf{dom}(D') \qquad (2(iv)) \square \end{aligned}$$

Proposition 4. $\forall i \in I.(D_i \overset{r}{\sqsubset} D'_i) \Rightarrow \biguplus_{i \in I} D_i \overset{r}{\sqsubset} \biguplus_{i \in I} D'_i$

Proof.

$$\begin{aligned} &\forall i \in I.(D'_i \rhd \{r\}) \uplus D_i = D'_i \\ &\Rightarrow \biguplus_i ((D'_i \rhd \{r\}) \uplus D_i) = \biguplus_i D'_i \\ &\Rightarrow \biguplus_i (D'_i \rhd \{r\}) \uplus (\biguplus_i D_i) = \biguplus_i D'_i \\ &\Leftrightarrow (\biguplus_i D'_i \rhd \{r\}) \uplus (\biguplus_i D_i) = \biguplus_i D'_i \qquad (2(ii)) \\ &\Leftrightarrow \biguplus_{i \in I} D_i \overset{r}{\sqsubset} \biguplus_{i \in I} D'_i \qquad (\text{def of } \overset{r}{\sqsubset}) \qquad \square \end{aligned}$$

The following establishes that computations over provenance tables preserves $\overset{r}{\sqsubset}$.

Proposition 5. $D \overset{r}{\sqsubset} D' \Rightarrow \hat{F}(D) \overset{r}{\sqsubset} \hat{F}(D')$

Proof. Assume premise.

$$\begin{aligned}
\hat{F}(D') &= \widetilde{F} \,;\, D' && \text{(def.)}\\
&= \widetilde{F} \,;\, (D' \rhd \{r\} \uplus D) && \text{(premise)}\\
&= (\widetilde{F} \,;\, (D' \rhd \{r\})) \uplus (\widetilde{F} \,;\, D) && (2(i))\\
&= ((\widetilde{F} \,;\, D') \rhd \{r\}) \uplus (\widetilde{F} \,;\, D) && (2(iii))\\
&= ((\hat{F}(D')) \rhd \{r\}) \uplus \hat{F}(D) && (\hat{F} \text{ def.})
\end{aligned}$$

Hence $\Rightarrow \hat{F}(D) \overset{r}{\sqsubset} \hat{F}(D')$ as required.

Definition 16 ((r,ε)-similarity for configurations). *We define the following similarity relations for (the components of) configurations:*

- $E \overset{r}{\sqsubset} E' \iff \forall tv.\, E(tv) \overset{r}{\sqsubset} E'(tv) \vee E(tv) = E'(tv)$
- $B \overset{r,\varepsilon}{\sqsubset} B' \iff B[r \mapsto \varepsilon] = B' \wedge \varepsilon \in [0, \mathscr{E}(r)]$
- $\langle P, E, B\rangle \overset{r,\varepsilon}{\sqsubset} \langle P', E', B'\rangle \iff P = P' \wedge E \overset{r}{\sqsubset} E' \wedge B \overset{r,\varepsilon}{\sqsubset} B'$

Finally, define $\mathbb{C} \overset{r,\varepsilon}{\sim} \mathbb{C}'$ *iff* $\mathbb{C} \overset{r,\varepsilon}{\sqsubset} \mathbb{C}'$ *or* $\mathbb{C}' \overset{r,\varepsilon}{\sqsubset} \mathbb{C}$.

The generalised version of Theorem 2 establishes the invariant relation between configurations from which the Theorem is a straightforward corollary.

Lemma 4. *If* $\sigma \overset{r}{\sim} \sigma'$ *and* $\mathbb{C}_0 \overset{\sigma}{\Rightarrow}_p \mathbb{C}$, *then* $\mathbb{C}_0 \overset{\sigma'}{\Rightarrow}_q \mathbb{C}'$ *where* $\mathbb{C} \overset{r,\varepsilon}{\sim} \mathbb{C}'$ *for some* ε *such that* $p \leq q.\exp(\mathscr{E}(r) - \varepsilon)$.

The proof is given in Appendix A.

6 Implementation and Experimental Results

In this section we start by briefly describing the implementation of ProPer. We continue with a small example to give a taste on how the tool is used, and finally we present some experimental results in terms of time and memory execution applied to a couple of benchmarks, comparing ProPer with PINQ.

6.1 Description of the Tool

We have implemented our PDP approach into the tool ProPer. The tool has been implemented in C#.

In order to interact with the tool, the user must be working on a programming environment (e.g., C#) and needs to create an instance of the ProPer class. After this, there are a number of constructs available thorough the ProPer API to initialise and manipulate data. ProPer is based on LINQ, and its interface has been designed having PINQ as inspiration so not surprisingly the way a user interacts with both tools is similar, modulo some syntactic differences.

Despite similarities there are important differences in the way PINQ and ProPer are implemented as explained below.

(i). When a new data set is given to ProPer, each record of the data set is assigned a unique key and an individual privacy budget. In PINQ records do not have a key and the privacy budget is global.
(ii). ProPer performs provenance tracking using the above mentioned record keys. One feature of this provenance tracking is that each record depends at most on one record from the input set.
(iii). Some transformations are implemented differently: In ProPer `Where` and `Select` are equipped with provenance tracking mechanisms, but this is not the case in PINQ.
(iv). Though the dynamic updating of databases (adding records) is possible in PINQ, the added records inherit the current global budget and thus they can be used as many times in queries as the old records. In ProPer, PDP guarantees that added records may participate in as many queries as their individual budget allows, not depending on others' record budget (or global budget).

In what follows we elaborate on how ProPer works. Let us assume the user has initialised a data set and wants to perform a number of transformations in order to make some queries. In order to do this, a ProPer object is first created, the data set is imported into ProPer, the transformations are applied to this object, and finally the user can then run arbitrary queries on that ProPer object. The above description is a simplification as ProPer only supports transformations where each resulting record depends on at most one record. If other transformations not respecting this constraint are neededthen a PINQ subroutine will be called, and treated like a primitive query.

In more detail, in ProPer, sensitive data is stored in a protected object with the generic type of `ProPer<T>` (in PINQ sensitive data is placed in a `PINQueryable<T>` object). When data is manipulated using $\uplus$-preserving transformations, provenance information is also transferred and attached to the resulting records. Two important supported transformations are

`Where` and `Select` representing the *projection*(π) and *selection*(σ) primitive operators in relational algebra. When it comes to aggregation, records with sufficient privacy budget are selected and their privacy budget is reduced. Also when a transformation with unsupported type of provenance is issued, the data set may switch to classical differential privacy by calling `AsPINQ(double epsilon)`. This reduces the budget of each involved record by ε, and creates a protected object of type `PINQueryable<T>`. From this point the resulting `PINQueryable<T>` object can be used in other arbitrary transformations defined in classical differential privacy or contribute in other ε-differentially private aggregations.

6.2 Example

As mentioned previously, PINQ introduces a special parallel composition operation for applying queries to disjoint parts of a data set. We argued that personalised budgets implies that the analyst does not need to construct parallel queries (§3) – it is just as efficient to pose sequential queries. But in situations where there is no natural partition of data our approach is not only more convenient, but also gives strictly better results. Let us assume a data set on which we will perform three queries, each one with accuracy ε, and such that the pairwise intersection of the intended domains of these queries is nonempty but where we know that that the intersection of the 3 queries is empty (see Figure 4). In the universe where each person is allowed to have at most two roles, the queries *Teachers*, *Managers* and *Headmasters* are asking about individuals in different roles. In this case, we cannot use parallel composition (as in PINQ) since it would require that the queries are disjoint. If we run them sequentially it would consume 3ε, whereas if we use PDP we will consume 2ε. (Note, that in the case of 3 disjoint queries, the PDP approach would consume the same budget as in the parallel case (ε).) The above case may be generalised: Given n queries such that each record is involved in at most g queries, would give the following: parallel composition (*à la* PINQ) cannot be applied, PINQ sequential composition would consume $n \cdot \varepsilon$, while PDP would consume $g \cdot \varepsilon$.

As a simple example we demonstrate how this analysis can be implemented in ProPer. We use the structure described below to store an individual's information, where each individual has at most two different roles: role1 (e.g., Teacher), and role2 (e.g., Manager).

```
public struct Individual
{
    public string name;
    public string role1;
    public string role2;
}
```

To initialise and populate our database with sample data we can pass an array of type `Individual` as an argument to the constructor method:

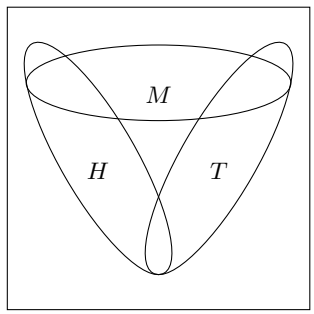

Fig. 4. T=*Teachers*, M= *Managers* and H=*Headmasters*

```
Individual[] population = new Individual[]
{
    new Individual{ name = "Alice",
                    role1 = "Teacher" ,
                    role2 = "Headmaster"},
    new Individual{ name = "Bob",
                    role1 = "Manager" ,
                    role2 = "Teacher"},
    ...
}
var protectData = new ProPer<Individual>
    (population.AsQueryable(),budget);
```

To construct a ProPer protected object that only stores information about Teachers we can use the *selection* operation. Note that each record in the resulting ProPer has a dependency relation with exactly one record from the input data set:

```
var Admins = protectData.Where(person=>(
    person.role1.Equals("Teacher") || person.role2.Equals("Teacher")
    ) );
```

Using the `Select` transformation we can modify an attribute's value or totally remove an attribute from a relation. For instance, as you can see in the following code, we transformed the table storing information about all Teachers into another table containing the length of their names:

```
var NameLengths = Admins.Select (person => (person.name).Length );
```

Finally, to extract information from the above table we call the method `AsPINQ(epsilon)` which will reduce the budget of each individual record from the data set under consideration by ε, and will create a `PINQueryable` object with total budget ε.

Now it is possible to run a classical differential privacy query; the simplest one being to call an aggregation function with accuracy ε, as shown in the code below.

```
Console.WriteLine (NameLengths.NoisyAverage(epsilon, x =>x /20) * 20
    );
```

Similarly, the ProPer object can be converted into a `PINQueryable` object as follows:

```
var pinqObj= NameLengths.AsPINQ(epsilon);
```

Note that `PINQueryable` objects can also be used in more complicated transformations like `Join` and `GroupBy`, available in PINQ.

As we have previously mentioned, another distinct feature of ProPer is its ability to deal with dynamic databases. For that, ProPer has the following available methods: `Update`, `Insert` and `Delete`. It is also possible for users to define their own methods to manipulate data if needed be. A function to refresh the contents of the database when it is called can be defined with the `SetRefresh()` method. The defined function is called by the client program each time `Refresh()` is called:

```
protectData.SetRefresh (obj=>(obj .Remove(predicate=>true))
     .Insert(newPopulation.AsQueryable() , epsilon)  );
protectData.Refresh();
```

6.3 Experimental results

To have a space and execution time comparison we implemented the *k-means clustering* algorithm both in PINQ and ProPer. This algorithm only uses *projection* and *selection* primitives which makes it a perfect candidate for comparison. The k-means algorithm accepts four parameters: a list of records, the number of centres, the number of dimensions, and the number of iterations. For the purpose of this research we fixed three of the parameters (Number of dimensions = 4 , Number of centres = 4 , Number of iterations = 5), and modified the number of records to see its effect on execution time and memory usage.

The result of our experiments concerning time is shown in Figure 5, where it is possible to see the effect on execution time when varying the number of records. As it can be seen in the figure, adding provenance tracking (ProPer) has a negative effect on the execution time and slows down the system by around 15 percent. Concerning the memory usage, ProPer implementation of the k-means clustering algorithm uses twice as much memory as the PINQ implementation. This can be motivated by the fact that each record has the type `double`, and for each record a key with type `int` and a structure to keep track of privacy budget is needed. This high memory usage is justifiable since these extra structures (key and privacy budget) has almost the same size as the size of the record. More generally the overhead in memory will be the ratio of the size of the record with and without the provenance record key.

6.4 Limitations

The restriction of ProPer to unary union-preserving transformations means that in some cases we simply have to fall back to using PINQ routines.

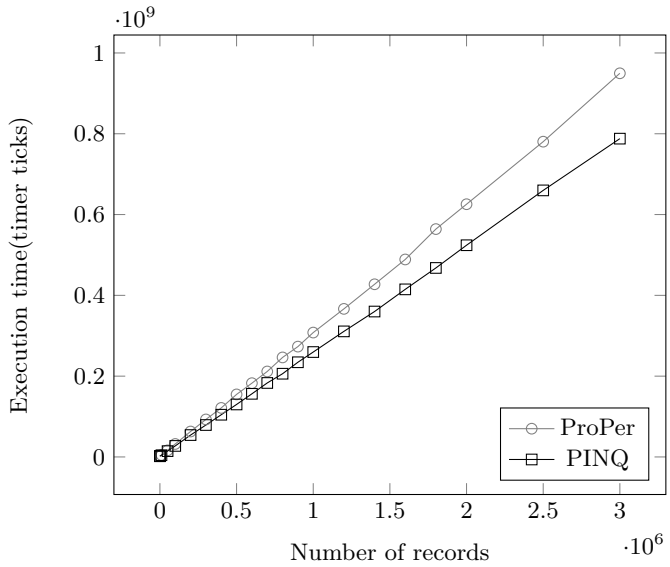

Fig. 5. Comparing PINQ and ProPer wrt execution time

For example, in the network analysis example of McSherry and Mahajan [16] the first transformation of the dataset is to group network requests by IP address. If the number of possible IP addresses was small and statically known, then we could iterate over this list to select the elements corresponding to each IP. But since they are not, the list of groups clearly has multiple provenances and is thus not supported by our method. So in this example we immediately fall back to using a PINQ subprogram, and appear to get no benefit from personalised budgets. But potential benefits from personalised budgets are not far away. For example, if one decided to restrict analysis to a specific geolocation, then personalised budgets would ensure that we don't waste the budget of the rest of the world. Another example is if traffic data arrives continuously over time, in which case we would automatically filter the records which have an exhausted budget without requiring any visible bookkeeping at the level of the analyst code.

7 Related work

The literature on differential privacy, although only ten years old, is already vast. For an overview we refer to Dwork's surveys [7, 6]. We consider work related along four dimensions: general systems which provide mechanisms for constructing differentially private analyses, approaches

to dealing with dynamic data, provenance and its connection to privacy, and approaches to improving the utilisation of privacy budget.

Systems Enforcing Differential Privacy As mentioned earlier, the interactive style of our system is inspired by PINQ [17]. The implementation of PINQ has a number of side-channels as pointed out in [12]; these and other implementation weaknesses ([19]) are certainly present in our implementation and it has not been our goal to focus on those. Other systems of note include Airavat [24], which is a MapReduce-based framework for constructing differentially private programs. Untrusted client analysts construct mappers, and the system provides differentially private reducers. Note that mapping is restricted in Airavat (by modifying the JVM) to be a union-preserving operation ("only a single record is allowed to affect the key-value pairs output by the mapper"), so it would be interesting to explore personalised budgets in that setting since it could potentially improve the budget utilisation over time, and perhaps remove the need to statically decide and enforce the exact number of elements produced by the mapper. Another line of work developing the Fuzz and DFuzz prototypes [23, 12, 10] describes non-interactive differential privacy in which the whole computation over private data is described by a domain-specific functional program, and a sensitivity-based static type system determines statically whether the computation will be within budget. This approach would combine well with ours by using it as a (necessarily side-effect free) query sublanguage.

Dynamic systems and data streams There are different senses in which a system might support dynamic data. In one sense the users are static, but their data arrive as a stream. One approach to privacy on streaming data is *Pan-privacy* [18], a stronger notion than differential privacy which ensures that the entire state of the system is private. Streaming PINQ is a version of PINQ that supports this kind of data [26]. GUPT introduces a novel concept of privacy which degrades over time. It seems that this is a feature that could be added to ProPer by periodic increase in the budgets of records.

Tschantz, Kaynar, and Datta [25] build a model and proof techniques for reasoning about interactive differential privacy with records that are input over time. They introduce a specific generalisation of differential privacy called differential noninterference. Their formal model of noninterference has similarities to ours, based on probabilistic transition systems. They develop bisimulation-like proof methods and similar ideas could be useful in our setting to refactor our main proof. In our setting the one system we reason about is parameterised over any program which uses the internal API. In the formal model of Tschantz *et al* the queries are supplied by the environment. Thus the model of the (malicious) analysis is the sequence of all possible queries (for all possible

input sequences). A question mark over this model is that is does not capture the *strategy* of the user; for a probabilistic systems it is known from the noninterference literature that modelling the user using nondeterminism rather than a strategy can hide the presence of information leaks [27, 20]. One aspect of their model is not captured in our system, they model bounded memory. This causes an unexpected magnification in the privacy cost of computations, since addition of one record into an input stream will cause a full memory to change by two records (the record itself and the record that it displaces).

Provenance and Lineage The notion of provenance that we use is more specifically called *why provenance* in the terminology of Cheney *et al* [2]. More specifically it is the *lineage* notion from Cui *et al* [3]. Union-preserving transformations are called *dispatchers* in that work.

Our main principle is the tracking of data from input to the point at which it is used in a query. Complementary to this, Birgisson, McSherry and Abadi [1] show how to improve privacy budgets by tracking from the result of a query to the final result of the (batch) program; if the query is not used to produce the final result then you don't need to count it's cost. In some sense this optimisation is already built into systems like Fuzz. Our model assumes that the results of intermediate queries are observed by the attacker, so using this principle would require us to refine our model, but it can be plugged straight into our implementation.

There are a number of other works which link the concepts of provenance and privacy, although these are mainly connected with answering queries about the provenance of data in a privacy preserving manner, e.g. [4].

Improving the Budgeting Accuracy of Composition Many methods in differential privacy deal with improving the ε-bound that is attributed to a given class of computations. Some of these are related to the deficiency of the sequential query composition principle, and are typically much more specialised (and therefore more technically sophisticated) than the method of provenance tracing described here – see for example [15, 14, 28]. Palamidessi and Stronati [21] provide a compositional method for improving the sensitivity estimation for relational algebra terms. It would be interesting to investigate whether these ideas can be used alongside our personalised approach.

Proserpio, Goldberg and McSherry [22] introduced wPINQ, a framework for weighted datasets. In wPINQ, "problematic" records (those inducing extra noise to preserve privacy) are specially treated. The idea is that in order to better preserve privacy while not degrading the accuracy of the query result, the weight of those individual records in an aggregate query is scaled down, instead of scaling up the noise added to all records. Note that weights in wPINQ are associated with each and every record,

whereas budgets in ProPer are associated just with individuals (the original inputs). By making weights part of every record, the privacy of the weights themselves will be protected by the requirements of the definition of differential privacy. Weights are used to track sensitivity at the level of each record level – similar to the fine-grained accounting achieved by tracking provenance. But the number of transformations that wPINQ supports is more than in PINQ. The price to pay is that every primitive query must be implemented to use the weights appropriately, and must be proven to be differentially private. In ProPer the correctness argument for the system itself has to be argued from first principles, but the method is able to reuse arbitrary differentially private queries as sub-programs without modification.

8 Conclusion

We have introduced a new concept of personalised differential privacy (PDP) that improves the bookkeeping regarding the cost of composed queries, and makes it easy to include dynamic expansion of the data base. We have realised this idea in the design of ProPer, a system which enforces PDP for all (deterministic) client programs that compute against a simple API. We have proved that the ProPer model provides personalised – and therefore also standard – differential privacy.

On Limitations of Affine Provenance In our development of this work, the implementation preceded the theoretical development [9]. Our first implementation traced the provenance for a much more general class of (SQL) functions, i.e. we traced provenance across operations like join, which implies non affine provenance (the provenance of a record may be more than one input record). Through our formalisation we subsequently discovered that this in fact violates differential privacy. Used as a descriptive mechanism, where we record privacy debt rather than spend from a budget, this approach is still sound since it never needs to exclude any records from queries, but it is less clear how to deploy such a system. The restriction to unary union-preserving functions, on the other hand, limits the functionality of client programs, but seems no worse than Airavat's mapper restrictions. In section 4.3 you can see a list of relational algebra operations that guarantee to have records with affine provenance. In any case, when a subcomputation cannot be expressed via a union-preserving transformation we can still plug in any other black-box differential privacy mechanism. This was further discussed in Section 6.4.

On Dynamic Data and Utility Perhaps the biggest advantage of PDP is that it smoothly supports dynamic databases in a PINQ style system – something that seems difficult to achieve in the presence of a single global

budget. Our prototype implementation shows a 15% slowdown compared to PINQ, requiring just a constant space overhead per record.

Another potential advantage of PDP is precisely personalisation; each individual can set her own privacy budget. However we are cautious in our assessment of this as a feature in its own right rather than just a means to an end. What is missing in the theory of PDP is a proper treatment of utility. The personal budget determines how quickly a record will be "used up". This complicates the understanding of the utility of the information returned by queries. But even if we start out with every record being assigned the same budget, if the analyst has no prior on the data then it can be hard to say much about the utility of any given query.

One particular case where utility guarantees may be easy to give (without a prior) is the case when the rate at which new data enters the database is sufficiently high relative to the rate at which queries consume their budgets. Assume a stream of inputs with flow rate f, queue size l and individual privacy budget of b. If we apply ε-differentially private queries at an execution rate below $\frac{b \times f}{l \times \varepsilon}$, then we can guarantee that ProPer can answer all queries without excessive noise caused by blocking too many old records from being used in queries. A more rigorous analysis of this idea is appropriate for future work.

Acknowledgements

This research has been partially supported by a grant from the Swedish Foundation for Strategic Research (SSF). Many thanks to our colleagues in the ProSec and Formal Methods groups for many helpful discussions, and special thanks Raúl Pardo Jiménez for participation in the early stage of the research, and to Niklas Broberg for comments on an earlier draft. Thanks also to Cédric Fournet and the anonymous referees for helpful comments.

References

[1] Arnar Birgisson, Frank McSherry, and Martín Abadi. Differential privacy with information flow control. In *Proceedings of the ACM SIGPLAN 6th Workshop on Programming Languages and Analysis for Security*, PLAS '11, pages 2:1–2:6. ACM, 2011.

[2] James Cheney, Laura Chiticariu, and Wang-Chiew Tan. Provenance in databases: Why, how, and where. *Found. Trends databases*, 1(4): 379–474, April 2009.

[3] Yingwei Cui and Jennifer Widom. Lineage tracing for general data warehouse transformations. *The VLDB Journal The International Journal on Very Large Data Bases*, 12(1):41–58, 2003.

[4] Susan B Davidson, Sanjeev Khanna, Sudeepa Roy, Julia Stoyanovich, Val Tannen, and Yi Chen. On provenance and privacy. In *Proceedings of the 14th International Conference on Database Theory*, pages 3–10. ACM, 2011.

[5] Cynthia Dwork. Differential privacy. In *33rd International Colloquium on Automata, Languages and Programming, part II (ICALP 2006)*, volume 4052, pages 1–12. Springer Verlag, 2006.

[6] Cynthia Dwork. A firm foundation for private data analysis. *Communications of the ACM*, 54(1):86–95, 2011.

[7] Cynthia Dwork. Differential privacy: A survey of results. In *Theory and Applications of Models of Computation*, volume 4978 of *Lecture Notes in Computer Science*, pages 1–19. Springer Berlin Heidelberg, 2008.

[8] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. Calibrating noise to sensitivity in private data analysis. In *Theory of Cryptography*, pages 265–284. Springer, 2006.

[9] Hamid Ebadi. PINQuin, a framework for differentially private analysis. Master's thesis, Chalmers University of Technology, 2013.

[10] Marco Gaboardi, Andreas Haeberlen, Justin Hsu, Arjun Narayan, and Benjamin C. Pierce. Linear dependent types for differential privacy. In *Proceedings of the 40th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '13, pages 357–370. ACM, 2013.

[11] Paul WPJ Grefen and Rolf A de By. A multi-set extended relational algebra: a formal approach to a practical issue. In *Data Engineering, 1994. Proceedings. 10th International Conference*, pages 80–88. IEEE, 1994.

[12] Andreas Haeberlen, Benjamin C Pierce, and Arjun Narayan. Differential privacy under fire. In *USENIX Security Symposium*, 2011.

[13] Ian Hayes. Multi-relations in z. *Acta Informatica*, 29(1):33–62, 1992.

[14] Georgios Kellaris and Stavros Papadopoulos. Practical differential privacy via grouping and smoothing. In *Proceedings of the 39th international conference on Very Large Data Bases*, PVLDB'13, pages 301–312. VLDB Endowment, 2013.

[15] Chao Li, Michael Hay, Vibhor Rastogi, Gerome Miklau, and Andrew McGregor. Optimizing linear counting queries under differential privacy. In *Proceedings of the Twenty-ninth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '10, pages 123–134. ACM, 2010.

[16] Frank McSherry and Ratul Mahajan. Differentially-private network trace analysis. *SIGCOMM Comput. Commun. Rev.*, 40(4):123–134, 2010. ISSN 0146-4833.

[17] Frank D McSherry. Privacy integrated queries: an extensible platform for privacy-preserving data analysis. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*, pages 19–30. ACM, 2009.

[18] Darakhshan Mir, S. Muthukrishnan, Aleksandar Nikolov, and Rebecca N. Wright. Pan-private algorithms via statistics on sketches. In *Proceedings of the Thirtieth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '11, pages 37–48. ACM, 2011.

[19] Ilya Mironov. On significance of the least significant bits for differential privacy. In *ACM Conference on Computer and Communications Security*. ACM, 2012.

[20] Kevin R. O'Neill, Michael R. Clarkson, and Stephen Chong. Information-flow security for interactive programs. In *CSFW*, pages 190–201. IEEE Computer Society, 2006.

[21] Catuscia Palamidessi and Marco Stronati. Differential Privacy for relational algebra: improving the sensitivity bounds via constraint systems. In *QAPL - Tenth Workshop on Quantitative Aspects of Programming Languages*, volume 85 of *Electronic Proceedings in Theoretical Computer Science*, pages 92–105. Open Publishing Association, 2012.

[22] Davide Proserpio, Sharon Goldberg, and Frank McSherry. Calibrating data to sensitivity in private data analysis. *40th International Conference on Very Large Data Bases, VLDB'14*, 7(8):637–648, 2014.

[23] Jason Reed and Benjamin C. Pierce. Distance makes the types grow stronger: A calculus for differential privacy. In *Proceedings of the 15th ACM SIGPLAN International Conference on Functional Programming*, ICFP '10, pages 157–168. ACM, 2010.

[24] Indrajit Roy, Srinath T. V. Setty, Ann Kilzer, Vitaly Shmatikov, and Emmett Witchel. Airavat: Security and privacy for mapreduce. In *NSDI*, pages 297–312. USENIX Association, 2010.

[25] Michael Carl Tschantz, Dilsun Kaynar, and Anupam Datta. Formal verification of differential privacy for interactive systems (extended abstract). *Electron. Notes Theor. Comput. Sci.*, 276:61–79, September 2011.

[26] Lucas Waye. Privacy integrated data stream queries. In *Proceedings of the 5th annual conference on Systems, programming, and applications: software for humanity*. ACM, 2014.

[27] J. Todd Wittbold and Dale M. Johnson. Information flow in nondeterministic systems. In *IEEE Symposium on Security and Privacy*, pages 144–161, 1990.

[28] Yonghui Xiao, Li Xiong, and Chun Yuan. Differentially private data release through multidimensional partitioning. In *Proceedings of the 7th VLDB Conference on Secure Data Management*. Springer-Verlag, 2010.

Appendix A

Proof of Main Lemma

Proof. Assume that $\sigma \overset{r}{\sim} \sigma'$ and $\mathbb{C}_0 \overset{\sigma}{\Rightarrow}_p \mathbb{C}$. We proceed by induction on the length of the trace σ, and by cases according to the last step of the trace.

Case 1: $\sigma = []$. Vacuous since we cannot have $\sigma \overset{r}{\sim} \sigma'$ if there are no inputs.

Case 2: $\sigma = \sigma_1 a$. Suppose that $\mathbb{C}_0 \overset{\sigma_1}{\Rightarrow}_{p_1} \langle P_1, E_1, B_1\rangle \overset{a}{\to}_{p_2} \langle P, E, B\rangle = \mathbb{C}$, and hence that $p = p_1 p_2$. We split this into two cases according to whether r is input on the last step, or earlier in the trace:

Case 2.1: $\sigma' = \sigma_1 a'$ **and** $\{a, a'\} = \{T, T \cup \{r\}\}$. Suppose that $a = T$ (the other case is argued similarly). Then we must have $P_1 \xrightarrow{tv:=\mathsf{input}} P$ for some tv, and hence:

$$\begin{aligned}\mathbb{C} &= \langle P, E_1[tv \mapsto \mathsf{Id}(T)], B_1[s \mapsto \mathscr{E}(s) \mid s \in T]\rangle \\ \mathbb{C}' &= \langle P, E_1[tv \mapsto \mathsf{Id}(T \cup \{r\})], B_1[s \mapsto \mathscr{E}(s) \mid s \in (T \cup \{r\})]\rangle \\ &= \langle P, E_1[tv \mapsto \mathsf{Id}(T \cup \{r\})], B_1[s \mapsto \mathscr{E}(s) \mid s \in T][r \mapsto \mathscr{E}(r)]\rangle\end{aligned}$$

Hence $\mathbb{C} \overset{r,\mathscr{E}(r)}{\sim} \mathbb{C}'$. Since the probability of the input transition is 1 we have $p = q$ and hence $p \leq q.\exp(\mathscr{E}(r) - \mathscr{E}(r))$ as required.

Case 2.2: $\sigma' = \sigma_1' a$ **and** $\sigma_1 \overset{r}{\sim} \sigma_1'$.

The induction hypothesis gives us q_1, P_1, E_1', B_1' and ε_1 such that

$$\mathbb{C}_0 \overset{\sigma_1'}{\Rightarrow}_{q_1} \langle P_1, E_1', B_1'\rangle \tag{3}$$

$$E_1 \overset{r}{\sim} E_1' \tag{4}$$

$$B_1 \overset{r,\varepsilon_1}{\sim} B_1' \tag{5}$$

$$p_1 \leq q_1.\exp(\mathscr{E}(r) - \varepsilon_1) \tag{6}$$

From here we argue by cases according to the rule applied for the last transition $\langle P_1, E_1, B_1\rangle \overset{a}{\to}_{p_2} \langle P, E, B\rangle$. In every case except for the query transition we will see that $p_2 = 1$, and that $\langle P_1, E_1, B_1\rangle \overset{a}{\to}_1 \mathbb{C}'$ for some $\mathbb{C}'$. In those cases it follows that $p \leq q \cdot \exp(\mathscr{E}(r) - \varepsilon)$ by taking $\varepsilon = \varepsilon_1$ and using (6).

Case 2.2.1: Input. In this case $a = T$ and $P_1 \xrightarrow{\mathsf{input}} P$. Hence we have transitions

$$\langle P_1, E_1, B_1\rangle \overset{T}{\to}_1 \mathbb{C}$$

$$\langle P_1, E_1', B_1'\rangle \overset{T}{\to}_1 \mathbb{C}'$$

where

$$\mathbb{C} = \langle P, E_1[tv \mapsto \mathsf{Id}(T)], B_1[s \mapsto \mathscr{E}(s) \mid s \in T]\rangle$$
$$\mathbb{C}' = \langle P, E_1'[tv \mapsto \mathsf{Id}(T)], B_1'[s \mapsto \mathscr{E}(s) \mid s \in T]\rangle$$

Since $r \notin T$, it follows easily from 4 and 5 that $\mathbb{C} \overset{r,\varepsilon_1}{\sim} \mathbb{C}'$.

Case 2.2.2: Silent. Similar (but simpler) argument to above – the only change in the configuration is the program component, so it follows directly from the IH.

Case 2.2.3: Constant Transformation. $P_1 \xrightarrow{tv:=T} P$ and hence $\langle P_1, E_1, B_1\rangle \xrightarrow{\tau}_1 \mathbb{C}$ and $\langle P_1, E_1', B_1'\rangle \xrightarrow{\tau}_1 \mathbb{C}'$ where

$$\mathbb{C} = \langle P, E_1[tv \mapsto \mathsf{Const}(T)], B_1\rangle$$
$$\mathbb{C}' = \langle P, E_1'[tv \mapsto \mathsf{Const}(T)], B_1'\rangle$$

and we reason as for case 2.2.1.

Case 2.2.4: Table variable. Similar to the previous case, $P_1 \xrightarrow{tv:=tv'} P$ and hence $\langle P_1, E_1, B_1\rangle \xrightarrow{\tau}_1 \mathbb{C}$ and $\langle P_1, E_1', B_1'\rangle \xrightarrow{\tau}_1 \mathbb{C}'$ where

$$\mathbb{C} = \langle P, E_1[tv \mapsto tv'], B_1\rangle$$
$$\mathbb{C}' = \langle P, E_1'[tv \mapsto tv'], B_1'\rangle$$

and we reason as for case 2.2.1.

Case 2.2.5: F-Transformation. Here $P_1 \xrightarrow{t:=F(t_1 \uplus \ldots \uplus t_n)} P$, and so we have

$$\mathbb{C} = \langle P, E_1[tv \mapsto \hat{F}(E_1(tv_1) \uplus \cdots \uplus E_1(tv_n))], B_1\rangle$$
$$\mathbb{C}' = \langle P, E_1'[tv \mapsto \hat{F}(E_1'(tv_1) \uplus \cdots \uplus E_1'(tv_n))], B_1'\rangle$$

Since, from 4 we have $E_1(tv_i) \overset{r}{\sim} E_1'(tv_i)$, $i \in \{1, \ldots, n\}$, and so from 5 and 4 it follows that $\hat{F}(E_1(tv_1) \uplus \cdots \uplus E_1(tv_n)) \overset{r}{\sim} \hat{F}(E_1'(tv_1) \uplus \cdots \uplus E_1'(tv_n))$ and hence we have that $\mathbb{C} \overset{r,\varepsilon_1}{\sim} \mathbb{C}'$.

Case 2.2.6: Query. Here we have a rule instance of the form

$$\text{Query}\frac{P_1 \xrightarrow{Q_\varepsilon(tv)?n} P}{\langle P_1, E_1, B_1\rangle \xrightarrow{n}_{p_2} \mathbb{C}}$$

and thus there is an analogous transition

$$\langle P_1, E_1', B_1'\rangle \xrightarrow{n}_{q_2} \mathbb{C}'$$

where

$$\mathbb{C} = \langle P, E_1, B\rangle \qquad B = B_1[s \mapsto B_1(s) - C(s) \mid s \in A]$$
$$\mathbb{C}' = \langle P, E_1', B'\rangle \qquad B' = B_1'[s \mapsto B_1'(s) - C'(s) \mid s \in A']$$

$$C = [s \mapsto \varepsilon \cdot \mathsf{size}(E_1(tv) \bullet \{s\}) \mid s \in \mathsf{dom}(B_1)]$$
$$C' = [s \mapsto \varepsilon \cdot \mathsf{size}(E_1'(tv) \bullet \{s\}) \mid s \in \mathsf{dom}(B_1')]$$

$$A = \{s \mid B_1(s) \geq C(s)\} \qquad L = E_1(tv) \bullet A$$
$$A' = \{s \mid B_1'(s) \geq C'(s)\} \qquad L' = E_1'(tv) \bullet A'$$
$$p_2 = \Pr(Q_\varepsilon(L) = n) \qquad q_2 = \Pr(Q_\varepsilon(L') = n)$$

Since the environments are unchanged in this transition, $E_1 \overset{r}{\sim} E_1'$ follows immediately from the induction hypothesis. Suppose, without loss of generality, that r is in $\mathbb{C}$. Then it remains to show that, for some ε',

$$B \overset{r,\varepsilon'}{\sqsubset} B' \tag{7}$$
$$p_1 \cdot p_2 \leq q_1 \cdot q_2 \cdot \exp(\mathscr{E}(r) - \varepsilon') \tag{8}$$

Let us first compare the respective cost mappings C and C': $E_1 \overset{r}{\sqsubset} E_1'$ gives $\mathsf{size}(E_1(tv) \bullet \{tv\}) = \mathsf{size}(E_1'(tv) \bullet \{s\})$ whenever $s \neq r$. Hence the only difference between C and C' is a single mapping:

$$C = C'[r \mapsto \varepsilon \cdot \mathsf{size}(E_1(tv) \bullet \{r\})]. \tag{9}$$

Now consider A and A'. Since C and C' only differ on r, then if $B_1(r) \geq C(r)$ then $A = A' \uplus \{r\}$. Otherwise $A = A'$. Consider these cases in turn:

Case 2.2.6.1: $A = A'$. and hence $L = L'$, and hence $p_2 = q_2$. By taking ε' to be ε_1, requirement (8) follows from the induction hypothesis (6). Since the budget of r is unchanged in either transition, then $B \overset{r,\varepsilon_1}{\sim} B'$ follows from the induction hypothesis (5).

Case 2.2.6.2: $A = A' \uplus \{r\}$.

$$\begin{aligned} L &= E_1(tv) \bullet (A' \uplus \{r\}) \\ &= E_1(tv) \bullet A' \uplus E_1(tv) \bullet \{r\} \\ L' &= E_1'(tv) \bullet A' \\ &= E_1(tv) \bullet A' \end{aligned}$$

where the last step follows since $r \notin A'$ and $E_1 \overset{r}{\sim} E_1'$. Hence $L = L' \uplus (E(tv) \bullet \{r\})$ – i.e. the difference in the size of the sets on which the respective queries are made is $\mathsf{size}(E(tv) \bullet \{r\})$. Since Q_ε is ε-differentially private, it follows from the definition of vanilla differential privacy that

$$p_2 \leq q_2 \cdot \exp(\varepsilon \cdot \mathsf{size}(E(tv) \bullet \{r\})) \tag{10}$$

Combining this inequality with (6) we get

$$p_1 \cdot p_2 \leq q_1 \cdot q_2 \cdot \exp(\mathscr{E}(r) - \varepsilon_1) \cdot \exp(\varepsilon \cdot \mathsf{size}(E(tv) \bullet \{r\}))$$

Rearranging the exponents gives $p_1 \cdot p_2 \leq q_1 \cdot q_2 \cdot \exp(\mathscr{E}(r) - \varepsilon')$ when $\varepsilon' = \varepsilon_1 - \varepsilon \cdot \mathsf{size}(E(tv) \bullet \{r\})$. We complete the proof by showing that this value of ε' gives $B \overset{r,\varepsilon'}{\sim} B'$. From the induction hypothesis (5) we have that B_1 and B_1' agree on all values in their domains except r, and from (9) we have that the same holds for C and C'. Thus B and B' only differ on r, for which

$$\begin{aligned} B(r) &= B_1(r) - C(r) \\ &= \varepsilon_1 - \varepsilon \cdot \mathsf{size}(E(tv) \bullet \{r\}) \end{aligned}$$

and hence $B \overset{r,\varepsilon'}{\sim} B'$ as required.

Paper III : Sampling and Partitioning for Differential Privacy

Privacy Security & Trust Conference, 2016

Sampling and Partitioning for Differential Privacy

Hamid Ebadi, Thibaud Antignac, and David Sands

Department of Computer Science and Engineering,
Chalmers University of Technology
Gothenburg, Sweden
{hamide,dave,thibaud.antignac}@chalmers.se

Abstract. Differential privacy enjoys increasing popularity thanks to both a precise semantics for privacy and effective enforcement mechanisms. Many tools have been proposed to spread its use and ease the task of the concerned data scientist. The most promising among them completely discharge the user of the privacy concerns by transparently taking care of the privacy budget. However, their implementation proves to be delicate, and introduce flaws by falsifying some of the theoretical assumptions made to guarantee differential privacy. Moreover, such tools rely on assumptions leading to over-approximations which artificially reduce utility. In this paper we focus on a key mechanism that tools do not support well: sampling. We demonstrate an attack on PINQ (McSherry, SIGMOD 2009), one of these tools, relying on the difference between its internal mechanics and the formal theory for the sampling operation, and study a range of sampling methods and show how they can be correctly implemented in a system for differential privacy.

1 Introduction

Sampling is a procedure in statistics that involves studying only a subset of a population to estimate properties and quantities of the entire population when access to the entire dataset is costly or practically infeasible. Several sampling methods have been introduced aiming to give statisticians an unbiased representative dataset model with properties generalisable to the entire population.

Regardless of the method, the main reasons to use sampling/partitioning, especially in the context of privacy preserving analysis, can be summarised in the following points.

- **Environment-driven.** Stream processing or real time systems usually suffer from limited computing and storage capabilities. Fitting data into memory is not always possible and the computation power is not always sufficient for real time response requirements.

To overcome these limitations, parallel and distributed computing (e.g. Map-Reduce) and computations on samples are frequently used.
- **Application-driven.** Machine learning algorithms [15] usually train an algorithm on a subset of a dataset called *training set* and then use the rest of the dataset to evaluate the effectiveness of the algorithm in the *validation/test* phase before it is used for the real world application. Keeping these two subsets separated is one of the main requirements otherwise it is not clear whether the algorithm generalizes or just remembers (memorises) the answers.
- **Utility-driven.** In the differential privacy setting, as explained in several papers [17, 6], when a dataset D is decomposed into arbitrary disjoint subsets D_i, the queries M_{ϵ_i} on these disjoint partitions provides less differential privacy cost than when these queries are executed in sequence. Sharing of a privacy budget between two analyses is an important outcome of parallel composition. This property, known as parallel composition, is particularly useful to efficiently build histograms.
- **Privacy-driven.** Intuitively, working on only a subset of sensitive data might result in less leakage of sensitive information. Random sampling has the potential to amplify privacy.

Studying the literature and tools that employ sampling and partitioning in differential privacy context we noticed informal, incomplete, and sometimes incorrect arguments on implication of sampling and partitioning on the privacy guarantees of an analysis. As an example we show that operators in PINQ are incorrectly handled and demonstrate an attack which reliably reveals the presence of an individual item (Section 3 and Appendix 1.1).

The main aim of this paper is to examine the range of sampling and partitioning operators supporting the mentioned goals and their differential privacy implications. We further study how PINQ and similar frameworks can be extended to support a range of operators for sampling and partitioning (Section 4).

Before we go further explaining how these operators positively or negatively affect the privacy of an analysis, let's have an overview on the sampling methods and the primary and differential privacy principles that PINQ and similar frameworks are built upon (Section 2).

2 Preliminaries

We begin by recapping the basic definition of differential privacy, and some standard terminology for sampling that we will use in the remainder of the paper.

Differential Privacy. Differential privacy is generally presented as a way to perturbate replies to queries made over a private dataset in such

a way that every individual can plausibly deny being a member of this dataset. The noise added to this effect has to be carefully controlled to reach an acceptable trade-off between privacy and utility preservation. This control is made through the parameter ϵ; hence the name of ϵ-differential privacy. The definition sets a bound on the ratio between the result of two queries made on datasets differing on only one element.

Definition 1 (Differential privacy). *Let two datasets A and A' be called* neighbours *if their symmetric difference is a singleton – i.e. $A = A' \cup \{a\}$ or $A' = A \cup \{a\}$ for some a,i.e., $|A \,\Delta\, A'| = 1$ (also known as the Hamming distance, where Δ is the symmetric difference operation). A randomized computation M provides ϵ-differential privacy if for any neighbours A and A' and any possible output $S \subseteq \mathrm{Rng}(M)$,*

$$\frac{\Pr[M(A) \in S]}{\Pr[M(A') \in S]} \leq e^{\epsilon}.$$

with $\mathrm{Rng}(f)$ *the range of a function f.*

This definition of differential privacy is the strongest among many variants, and is the version studied in this paper. It is worth mentioning one close relative, which also goes by the same name but is subtly different. The ***bounded differential privacy*** [14] is obtained by considering two datasets A and A' to be neighbours if one can be obtained from the other by *changing* exactly one element. The general relation between the two definitions is that if M is ϵ-differentially private then it will be (at least) 2ϵ-bounded differentially private. But the converse does not hold, since differential privacy protects small changes in the size of the dataset, whereas bounded differential privacy does not.

The use of Definition 1 is important in the design of tools such as PINQ because queries target not only the input dataset but also datasets constructed by transforming the input with relational algebra operations. Bounded differential privacy does not compose well with many standard and useful data transformations. To see this we make the following observation: the size operation which (deterministically) returns the size of its dataset argument is 0-bounded differentially private. I.e., you can return the size of the dataset at zero privacy cost. But suppose that the dataset was built by first selecting all individuals with a specific social security number. Under (strong) differential privacy this is not a problem since selection preserves the neighbourhood property. However, it does not preserve the corresponding neighbourhood relation under bounded differential privacy. This behaviour cannot be allowed because the size operation would reveal the presence of a particular individual.

Definition 2 (Function sensitivity). *Function sensitivity is the maximum change of a function's resulting value caused by any single item change. For a function $f\colon \mathcal{A} \to \mathbb{R}$ and any two neighbouring datasets*

$A, A' \subseteq \mathcal{A}$, *sensitivity is defined as* $\text{sensitivity}(f) = \max_{A,A' \subseteq \mathcal{A}} |f(A) - f(A')|$.

Compositions. Many statistical analyses can be expressed as a chain of relational algebra transformations of some input data followed by an aggregation function. *Stability*, closely related to sensitivity, is a measure of how much a data transformation might scale up the sensitivity of a query.

Definition 3 (Transformation stability). *A transformation T, function from datasets to datasets, is said to be c-stable if for any two neighbouring datasets A and A', we have $|T(A) \, \Delta \, T(A')| \leq c$.*

The result of composing a function f with a c-stable transformation has the sensitivity of $c \times \text{sensitivity}(f)$. As a result the composition theorem of differential privacy is immediate.

Theorem 1 (Transformation composition [17][Theorem 2). *] Let M provide ϵ-differential privacy, and T be an arbitrary c-stable transformation. The composite computation $M \circ T$ provides $\epsilon \times c$-differential privacy.*

The relation between stability and differential privacy parameters lies at the heart of the formal theory behind differential privacy tools. These tools take advantage of some simple composition properties of queries themselves, namely that composite queries have additive privacy cost – or even better when the data is partitioned into disjoint sets.

Theorem 2 (Sequential composition). *Composition of n queries $M_1, M_2, \ldots, M_n$ that are executed sequentially can be seen as one query with $\sum_{1 \leq i \leq n} \epsilon_i$-differential privacy.*

A "better" result [17, Theorem 4] is obtained if queries are executed on disjoint subsets (partitions) of the dataset. In that case the differential privacy bound of $\max_{1 \leq i \leq n}(\epsilon_i)$ is achieved. We will state this in a more general form, but to do so we need to lift the definition of differential privacy to pairs of datasets, by defining (D_1, D_2) and (D_1', D_2') to be neighbours if D_1 and D_1' are neighbours and $D_2 = D_2'$ or vice-versa (see [23]). If M_1 and M_2 are queries on D, let $\langle M_1, M_2 \rangle$ denote the parallel query which when given a pair of datasets, returns a pair of query results, i.e., $\langle M_1, M_2 \rangle (D_1, D_2) = (M_1(D_1), M_2(D_2))$. Now the parallel composition can be stated as follows:

Theorem 3 (Parallel composition). *If M_i is ϵ_i-differentially private, $i \in \{1, 2\}$, then $\langle M_1, M_2 \rangle$ is $\max(\epsilon_1, \epsilon_2)$-differentially private.*

Note that the parallel composition theorem as we have stated it, unlike [17, Theorem 4], does not mention partitioning; the benefit of the theorem

comes by creating the input to $\langle M_1, M_2 \rangle$ by a partitioning operation. Also, not all strategies for partitioning are useful here. Indeed, the key is to have a partitioning operation of stability one, and thus we can apply the transformation composition theorem favourably. Fortunately, partitioning using a fixed partition on the whole domain has this property. The potential benefit of parallel composition motivates our study not only of random sampling but also of random partitioning.

Sampling. In the following, we will consider a variety of data sampling methods, so we also fix the terminology we will use. A *sampling method* is a probabilistic function which, given an input set D, produces a multiset over D, known as a sample. This very general concept includes deterministic sampling operations (e.g., where D is an indexed set, and the sample is a fixed number of elements based on the index) – for example what is sometimes called *convenience sampling* which simply selects the most conveniently accessible elements.

It also includes the case where the sample is a simple subset of D – and in this case the sampling method is said to be *without replacement.*

Definition 4 (Uniform sampling). *If RS is a sampling method, define $\overrightarrow{RS(D)}$ to be the set of possible outcomes – i.e. $\{K \mid \Pr[RS(D) = K] > 0\}$ Then RS is a uniform sampling method if the probability of obtaining any two samples in $\overrightarrow{RS(D)}$ is the same – i.e. the possible samples are all equally likely.*

Uniform sampling includes the standard notion of *simple random sampling*, with or without replacement, as well as fully deterministic sampling methods.

Sampling methods are invariably parameterised over the desired "size", which indicates, for example, the number of elements to be taken in the sample, or the probability that each element is selected. As a consequence, strictly speaking, we will be dealing with families of sampling functions $\{RS_k\}_{k \in \text{size}}$.

Note that any sampling method without replacement can be viewed as a partition of the input into two disjoint subsets, the sample and the rest.

We are now ready to have a look at the state of affairs to perform sampling in the wild and to develop the limitations mentioned in introduction.

3 Sampling in PINQ

The basic principles highlighted in the theorems of the previous section, and generalisations thereof, form a basis for general purpose programming frameworks that allow construction of analyses which are differentially private by construction [17, 26, 9, 10, 18, 22]. These frameworks

use a similar method to construct compound queries, but for the purpose of this paper we focus on PINQ [17].

This tool is a platform to perform differential privacy-preserving analysis on datasets. It is based on the LINQ interface from the Microsoft .NET framework. Differential privacy properties are automatically and transparently enforced by PINQ when the naive user only expresses the queries of interest.

The aim in this section is to analyse the status of sampling in PINQ to motivate the extension of the framework with sampling and partitioning operations. As mentioned, PINQ is an API for enforcing differential privacy, and is built from deterministic transformations T_i, which are variants of the well-known SQL-like operations, and primitive differentially-private aggregation operations M_ϵ, parameterised by the intended privacy level ϵ. To get an idea of the way PINQ works, suppose that we take the original data, perform a sequence of transformations $T_1, \ldots T_n$ on it, apply a primitive aggregation operation M_ϵ, and publish the result. PINQ counts the overall privacy cost of this by deducting $\epsilon \times \Pi_{i \in \{1,..n\}}$ stability(T_i) from the budget (if the budget is insufficient, an exception is thrown).

Take and Skip. Since the theory of PINQ is based on non-probabilistic transformations, PINQ supports just two sampling operations which allow the selection of (at most) a fixed number of elements based on the underlying ordering of the elements in the database.

PINQ supports sampling as a stand-alone transformation but not with the purpose of amplifying privacy. The two complimentary transformation methods `Take(n)` and `Skip(n)` are based on their equivalent methods in LINQ. `Take(n)`, as explained in Microsoft Software Developer Network documentation [2] "returns a specified number of contiguous elements from the start of a sequence" and `Skip(n)` [1] "bypasses a specified number of elements in a sequence and then returns the remaining elements". Since these two methods work similarly, albeit in a dual manner, we only discuss the issue with the `Take(n)` method in this paper. A similar argument holds for the `Skip(n)` method.

The sensitivity of Take. Discussing the sensitivity of Take is tricky because it treats the data A as a sequence and not a set. In the following we assume that the index is part of the data. Let Take(A, n) be the mathematical function corresponding to the `A.Take(n)` PINQ operation. When `n` $\leq |$`A`$|$ the difference between results of Take(A, n) and Take$(A \cup \{a_x\}, n)$ is assumed to be zero, and when $n > |A|$, the difference is one as it returns all the elements. Consequently, in PINQ, the stability of Take transformations is assumed to be one.

However, the ordering of elements influences the stability. As an example, in the case of union of item a and dataset A, there is no guarantee that element a takes the highest ordering value and is placed at the end of

the list A. In the current implementation the ordering of resulting dataset follows the following rules:

- the relative ordering of items within a set remains the same, and
- the order of sets passed as arguments to union determines the order of sets.

To be more specific, when a_x is added to the end of the list (the item a_x appears as the second argument) we have a stability of zero: $\text{Take}(\{a_1 \dots a_n\}, 1) \ \Delta \ \text{Take}(\{a_1 \dots a_n\} \cup \{a_1\}, 1) = \{a_1\} \ \Delta \ \{a_1\} = \emptyset$. When the item is added to the beginning of the list we observe a stability of two:
$\text{Take}(\{a_1 \dots a_n\}, 1) \ \Delta \ \text{Take}(\{a_x\} \cup \{a_1 \dots a_n\}, 1) = \{a_1\} \ \Delta \ \{a_x\} = \{a_1, a_x\}$. This demonstrates that PINQ's assumption that Take has stability of 1 is incorrect. The problem with the incorrect sensitivity of Take is not immediately visible when applying aggregation operations to the result. This is because many of the aggregation operations supported by PINQ, e.g. `NoisyCount`, provide the same ϵ of privacy for both variants of differential privacy, since they are no more sensitive to removing an element (a symmetric difference of one) than they are to changing an element (a symmetric difference of two).

An attack that can scale up the difference by any arbitrary factor is demonstrated and explained in Listing 1 from Appendix 1.1. The proof of concept code determines the existence of a specific element (i.e., number 7) in the dataset with higher probability than the limited privacy bound ϵ. The idea behind the proof of concept is to apply some transformation that iteratively adds random items to the dataset if an individual item, here number 7, is not present in the dataset. With an actual dataset of size 9, we got 9867.4529 ($\approx 10000 + 9$) as the randomized size of the protected dataset object in one experiment after 10000 iterations. Running the same program, by adding the number 7 results in -20.0768 (≈ 9) as the randomized size. The large distance between the two possible outcomes can reveal the presence of any item in the database.

4 Uniform Sampling and Partitioning

The previous section highlights some shortcomings in PINQ:

- sampling for the purpose of scaling data to available resources is the only supported operation, and
- the privacy cost (the sensitivity) is worse than assumed, and thus taking elements halves the available privacy budget.

In this section we consider a variety of sampling methods with a view to including them in a PINQ-like framework. We wish to explore sampling for the various purposes listed in the introduction, and answer the following questions:

1. From a privacy perspective, can we do (significantly) better than Take for selecting a specific number of elements by using random sampling?
2. Can we partition the unknown-sized data into portions of known relative size, to share the data between different analyses and to take advantage of the parallel composition benefits of analyses on disjoint data?
3. For which kinds of sampling can we achieve amplification of privacy?
4. Is it possible to do partitioning that is parameterised on the size of the intended sample?
5. What are the stability properties of a well-behaved partitioning transformation?

Methods that are used for sampling data from a dataset is generally categorised into fixed size and unfixed size sampling methods. Fixed size sampling methods, given the sample size as a numerical number or a fraction of dataset, return a fixed number of elements every time the sampling is performed. Common fixed size sampling methods, including sampling without replacement, sampling with replacement, and fraction sampling are studied in this paper. For the unfixed size sampling we study Bernoulli sampling in which, given the sample inclusion probability, each element of the population is subject to an independent Bernoulli trial which decides about its inclusion.

Random uniform sampling *with* and *without* replacement are used when the algorithm, the system capacity, or the real-time requirements enforces a specific number of items for the system input (which are all environment-driven limitations). Randomized fraction sampling guarantees a precise ratio between the sample size and the population. Knowing the population size, one can compute the sample size. However when the size of population is not known (in the case of differential privacy) the sample size remains unknown. The size of samples from Bernoulli sampling is dynamic and follows the Bernoulli distribution. All the sampling methods can be used to reduce the size of the database but it is the application that influences the choice of the sampling method. In the following, we focus on utility and privacy-driven goals. Before this, we need to revisit the notion of stability.

4.1 From Deterministic to Probabilistic Transformations

In this subsection we look at general considerations in the analysis of sampling methods. We generalise the notion of stability and the associated composition methods from the deterministic to the probabilistic case, introduce random partitioning, and outline a simple strategy for reasoning about the differential privacy of queries applied after sampling.

Probabilistic Stability. The definition of transformation stability (Definition 3) is not useful when lifted naively to non-deterministic transformations, since in the worst case the Hamming distance between two samples of size n is $2n$ when the samples are selected from two disjoint subsets. While the stability of $2n$ is an obvious upper-bound for PINQ's randomized `Take(n)` method, we investigate the possibility of introducing a lower upper bound that can be plugged into differential privacy frameworks.

To introduce uniform sampling, we propose a generalisation for definition of stability that allows us to reason about uniform sampling and its composition with differential private mechanisms. This generalised notion of stability has to be independent of the content and of the size of the database and to be pluggable as a function of the privacy cost in these frameworks.

Definition 5 (Probabilistic stability). *Let σ be a function in $\mathbb{R}^{\geq 0} \to \mathbb{R}^{\geq 0}$. We say a randomized transformation RS is σ-probabilistically stable if for any ϵ-differentially private mechanism M, $M \circ RS$ is $\sigma(\epsilon)$-differentially private.*

We observe that c-stability can be expressed as a σ-probabilistic stability as follows:

Proposition 1. *If T is c-stable, then T is σ-probabilistically stable with $\sigma(\epsilon) = c \times \epsilon$.*

Thus this simple generalisation provides a drop-in replacement for the composition principle of Theorem 1 since it is compositional by construction; the simple accounting mechanism of PINQ can be generalised from deterministic transformations with known c-stability, to randomised transformations with known σ-probabilistic stability:

Proposition 2. *Suppose that $RS_1, \ldots, RS_n$ is a sequence of randomised transformations for which RS_i is σ_i-probabilistically stable, and M is ϵ-differentially private. The mechanism which computes $M \circ RS_1 \circ \cdots \circ RS_n$ is ϵ'-differentially private, where $\epsilon' = (\sigma_1 \circ \cdots \circ \sigma_n)(\epsilon)$.*

This follows easily from Proposition 1, the associativity of function composition, and a simple induction on n.

Partitioning and Parallel Queries. The sampling algorithms that are considered in this paper (except sampling with replacement), can also be seen as a partitioning algorithm. Selected items for the sample and the excluded items construct the two partitions that cover all the items.Let RS be a sampling method without replacement. Let $\hat{RS}$ denote the partitioning operation defined as

$$\Pr[\hat{RS}(D) = (K, \bar{K})]$$

$$= \begin{cases} \Pr[RS(D) = K] & \text{if } K \cup \bar{K} = D \text{ and } K \cap \bar{K} = \emptyset \\ 0 & \text{otherwise} \end{cases}$$

It turns out that for just one sampling method we get a low σ-sensitivity (i.e. where $\sigma(\epsilon) < \epsilon$, but for others the results are more expansive ($\sigma(\epsilon) > \epsilon$).

Reasoning About Sampling Methods. Probabilistic stability measures the effect of a transformation on the privacy that a differentially private algorithm enjoys. We observe that any transformation that is expressed in a primitive recursive form keeps the amount of changes limited in each iteration, and have a bounded and easily understood stability. This is the case, for example, for *map* functions (such as the projection operation of relational algebra), recursively defined for some mapping function f as $\text{map}_f\,(D \cup \{a\}) = \text{map}_f\,(D) \cup \{f(a)\}$. In this form it is immediate to see that it has stability of one. In general a recursive equation which allows us to compare the operator's action on $D \cup \{a\}$ and D respectively is precisely what we need to reason about (probabilistic) stability.

Thus, in the following subsections we take the same approach for reasoning about sampling. It turns out that we have a useful generic theorem for recursively specified sampling:

Theorem 4. *Let RS be a uniform sampling function. If RS is characterised by a recursive equation $RS(D + \{e\}) = \psi_D(RS(D), e)$ for some randomised function ψ_D then:*

- *The probabilistic stability of uniform sampling RS is*

$$\sigma(\epsilon) = \ln \sum_{i \in \mathbb{N}} e^{i\epsilon}.\, \Pr[|\psi_D(K, e)\ \Delta\ K| = i].$$

- *The probabilistic stability of uniform partitioning $\hat{RS}$, built based on uniform sampling RS, is*

$$\hat{\sigma}(\epsilon) = \ln \sum_{i \in \mathbb{N}} e^{(i+|i-1|)\epsilon}.\, \Pr[|\psi_D(K, e)\ \Delta\ K| = i].$$

The proof of this theorem is given in Appendix 1.2.

These theorems extract the common part of the reasoning about sampling methods in this paper. But how easy is it to find a (nontrivial) recursive form for a sampling algorithm? Some methods are trivial to phrase in this form (e.g. Bernoulli sampling, for example). For others we can find a recursive form by looking to the literature on *online* (stream) versions of sampling methods. In a streaming version of a sampling method a new sample is computed from the previous sample and the newly arrived datum. As a consequence, this can be viewed as a recursive specification of the underlying sampling method.

4.2 Uniform (Fixed Size) Sampling Without Replacement

The first method we consider is fixed size uniform sampling without replacement which we denote by the size-indexed family of functions $\{R_n\}_{n\in\mathbb{N}}$. This method is potentially useful when a specific maximum sized target set is desired. Let n be the (maximum) number of elements to be sampled, and D the dataset. When $n < |D|$, n non-distinct items are randomly chosen from the dataset D. When $n \geq |D|$ the entire dataset D is returned. Each item can be selected at most once, which means a sample larger than the dataset itself is not possible.

Definition 6 (Probabilistic behaviour of R_n). *If K is any random sample in the range of $R_n(D)$ (shown as $K \in \overrightarrow{R_n(D)}$, the probabilistic behaviour of random uniform sampling without replacement when $n < |D|$ is* $\Pr[R_n(D) = K] = \binom{|D|}{n}^{-1}$. *When $n \geq |D|$, the probability doesn't play any role and the function behaves deterministically as:* $\Pr[R_n(D) = D] = 1$.

This follows from the simple fact that R_n is a uniform sampling method, so the probability of any sample is just one over the number of possible samples of that size.

One common implementation of sampling without replacement is based on the Fisher-Yates shuffle [8] with the complexity of $O(n)$ to randomly pick n indices from the array that is holding the item set. This algorithm relies on the presence of all elements in the system and knowing the exact size of the dataset. However in the streaming setting where the size of dataset is not known in advance or when the dataset is larger than the available memory, Vitter [29] introduced the *Reservoir sampling algorithm* with a complexity of $O(|D|)$. An inductive proof for this algorithm can be found in [3] and [28].

The simplified Algorithm 1, known as reservoir sampling, produces a sample with the required probabilistic property. In each iteration the algorithm maintains a reservoir sample that keeps the sample collected so far. When a new data item is introduced, the algorithm inductively construct the new sample by using the sample constructed from the previous step [4].

The algorithm can be explained as follows. For the first n items introduced (one by iteration), the items are directly put into the reservoir. Once the reservoir is full and cannot get the i^{th} item ($i > n$), the algorithm generates a random number r between 1 and i. If r is less than or equal to n, it replaces the element stored in the index r of the reservoir with the new item. This determines whether an item should be included or excluded.

The array D in the Algorithm 1 can be a stream of data items processed one at the time (D[i]). This algorithm provides a recursive specification of random sampling without replacement with the base

Algorithm 1 Uniform Sampling Without Replacement

```
function R_n(D)
    for i ← 1, |D| do
        if i ≤ n then
            K[i] ← D[i]                          ▷ Deterministically add D[i]
        else
            r ← RANDOM(1, i)
            if r ≤ n then
                K[r] ← D[i]                      ▷ Include D[i]
                                                 ▷ Otherwise exclude D[i]
    return K[1..n]
```

case $R_n(D) = D$ when $|D| \leq n$ and the recursive case in the form of $R_n(D \cup \{e\}) = \psi_{n,|D|}(R_n(D), e)$. The family of probabilistic functions $\psi_{n,|D|}$ models inner iterations of the Algorithm 1. When the reservoir is not full ($i \leq n$) all items are added to the reservoir deterministically. For $i > n$, the incoming element replaces one item of the reservoir with the probability of $\frac{n}{|i|}$.

Theorem 5. *The following is a recursive characterisation of R_n:*

$$\begin{array}{ll} R_n(D) & = D \qquad\qquad\quad \textit{if } |D| \leq n \\ R_n(D \cup \{e\}) & = \psi_{n,|D|}(R_n(D), e) \quad \textit{otherwise} \end{array}$$

such that for all samples K of D of size n, $\psi_{n,|D|}$ is the probabilistic function defined by the following:

$$\begin{aligned} \Pr[\psi_{n,m}(K, e) = K] &= \tfrac{m+1-n}{m+1} \\ \Pr[\psi_{n,m}(K, e) = K^{+e}_{-f}] &= \tfrac{1}{m+1} \quad (\textit{for any } f \in K) \\ \Pr[\textit{otherwise}] &= 0 \end{aligned}$$

where K^{+e}_{-f} stands for the dataset K to which the element e has been added and the element f removed.

A proof of this theorem is provided in Appendix 1.4.

Theorem 6 (Probabilistic stability of R_n). *Random sampling of n elements without replacement, R_n, has a probabilistic stability $\sigma(\epsilon) = \ln(\frac{ne^{2\epsilon}+1}{n+1})$.*

The proof is outlined as follows. To find this probabilistic stability σ for sampling without replacement, we instantiate Theorem 4 for different value of D:
Case $n \geq |D \cup \{e\}|$: using $\psi_{n,|D|}$ from Theorem 5 we have the upper bound of $\sigma(\epsilon) = \epsilon$.
Case $n < |D \cup \{e\}|$: using $\psi_{n,|D|}$ from Theorem 5 we have the upper-bound of $\ln(\frac{ne^{2\epsilon}+1}{n+1}) < 2\epsilon$.

The maximum value appears in the case $n < |D \cup \{e\}|$. For practical purposes (small ϵ, large n), the value of $\ln(\frac{ne^{2\epsilon}+1}{n+1})$ doesn't lead to a

significant improvement of the privacy cost compared to the deterministic case 2ϵ.

Partitioning property. We now consider the case of a parallel query of a partition using R_n.

Theorem 7. *Let $\hat{M} = \langle M_1, M_2 \rangle$ where M_1 and M_2 are ϵ-differentially private queries. Then $\langle M_1, M_2 \rangle \circ \hat{R}_n$ is $\hat{\sigma} = 3\epsilon$-differentially private.*

The proof is as follows: consider the two possible cases and, by using Theorem 4:
Case $n \geq |D| + 1$: $\hat{\sigma(\epsilon)} = \epsilon$.
Case $n < |D| + 1$: $\hat{\sigma}(\epsilon) = \ln(\frac{ne^{3\epsilon}+1}{n+1}) < 3\epsilon$.

In total we have a probabilistic stability of $\hat{\sigma}(\epsilon) = \ln(\frac{ne^{3\epsilon}+1}{n+1}) < 3\epsilon$.

4.3 Uniform (Fixed Size) Sampling with Replacement

This method of sampling takes n elements from the dataset D. Each selection is independently done from the entire dataset D, meaning an item may appear more than once in the sample or the sample may be larger than the dataset itself. We have analysed this sampling method using ideas from Park et. al [21, 20] who propose several algorithms with different complexities and prove their correctness. The algorithm closest to the recursive form is the *RSWR-naive* algorithm [21]. We have been able to show that the probabilistic stability of the recursive step is $\sigma(\epsilon) = \ln \sum_{0 \leq l \leq n} e^{2l\epsilon} \cdot \binom{n}{l} (\frac{1}{n})^l (\frac{n-1}{n})^{n-l}$. We omit the details since the result is not really useful; it is easy to see that we have a lower bound of $n \times \epsilon$ which is already not practical; this bound is achieved by comparing the two neighbours $\emptyset$ and $\{a\}$. In the former case we must get an empty sample, but in the latter case a sample of n copies of a.

4.4 Fraction Sampling

Fraction sampling randomly chooses precisely $\lfloor |D| \times p \rfloor$ items (with $\lfloor . \rfloor$ being the floor function) from dataset D, which corresponds to a fraction of items of approximately p. While in statistics this method and fixed size sampling are treated similarly (a sample size can be expressed as a fraction/ratio), the strong notion of differential privacy where the exact size of sample is secret makes this distinction useful in our case. The advantage of using this method of sampling over fixed size sampling is that the analyst doesn't need to have any information about the number of elements in the database while still getting precise partitioning.

One can see that fraction sampling, which samples a fraction p of elements from the dataset, is similar to sampling without replacement in which the size of reservoir is dynamic and increases over time. *Pareto πps schema* – or *order sampling schemes* – introduced by Rosén [24, 25] is referred to as the class of algorithms for sampling with probability proportional to size (πps). The basic form of these algorithms, with equal

inclusion probability, assigns each item a unique random value chosen from a certain range (0,1) and calculate the Pareto ranking variable that is basically just an artificial random ordering. A similar algorithm for a distributed setting that can also be used to achieve fraction sampling is explained in [5]. The authors state that:

> "The core insight behind reservoir sampling is that picking a random sample of size k is equivalent to generating a random permutation (ordering) of the elements and picking the top k elements."

For the purpose of this research, centralised system with equal probabilities for each item, the algorithm can be simplified as Algorithm 2. We claim that fraction sampling, P_p, with $K \in \overrightarrow{P_p(D)}$, $lb = \lfloor p.|D| \rfloor$ (the

Algorithm 2 Fraction Sampling

```
function P(D, p)
    for i ← 1, |D| do
        r ← RANDOM(1, i)
        R[i] ← R[r]
        R[r] ← D[i]
    return R[1..⌊p × |D|⌋]
```

current sample size) and $ub = \lfloor p.(|D|+1) \rfloor$ (possibly the new sample size after addition of e), has the following recursive characterisation:

$$\begin{aligned} P_p(\emptyset) &= \emptyset && \text{(base case)} \\ P_p(D \cup \{e\}) &= \psi_{p,D}(P_p(D), e) && \text{(recursive step)} \end{aligned}$$

if K is a valid sample of $P_p(D)$, for the case $ub - lb = 1$ when we increase the size of reservoir, the probabilistic function $\psi_{p,D}$ is defined as

$$\Pr[\psi_{p,D}(K, e) = K \cup \{e\}] = \tfrac{1}{|D|+1}$$

$$\Pr[\psi_{p,D}(K, e) = K \cup \{e\} \setminus \{f\} \cup \{x\}] = \tfrac{ub}{|D|+1}$$

$$\Pr[\psi_{p,D}(K, e) = K \cup \{x\}] = 1 - \tfrac{ub+1}{|D|+1}$$

For the case $ub - lb = 0$, when we have no change in the size of reservoir, it is defined as:

$$\Pr[\psi_{p,D}(K, e) = K] = 1 - \tfrac{lb}{|D|+1}$$

$$\Pr[\psi_{p,D}(K, e) = K \cup \{e\} \setminus \{f\}] = \tfrac{lb}{|D|+1}$$

Theorem 8. *The probabilistic stability of fraction sampling, P_p, is* $\sigma(\epsilon) = \ln(\max(e^{2\epsilon}p + 1 - p, e^{3\epsilon}p + e^{\epsilon}(1-p)))$.

The probabilistic stability is the maximum of the upper bound for the case $(ub - lb = 1)$ and the case $(ub - lb = 0)$.

Partitioning Property. Similarly considering the maximum of both cases, we get a probabilistic stability of $\hat{\sigma}(\epsilon) = \ln \max(e^{3\epsilon}p + (1-p)e^{\epsilon}, e^{5\epsilon}p + 1 - p)$.

4.5 Bernoulli Sampling

The differential privacy of Bernoulli Sampling has been previously considered in the literature [27, 16] with the aim of amplifying privacy. Bernoulli Sampling independently decides about inclusion of each item from the population. Given inclusion probability of b, iterating over all members of a population, Bernoulli trials with probability b decides about the inclusion of each item.

Definition 7. *In Bernoulli sampling, a sample $K \subseteq D$ of size n has a probability of* $\Pr[B_b(D) = K] = b^n(1-b)^{|D|-n}$.

The following is a recursive characterisation of B_b:

$$\begin{aligned} B_b(\emptyset) &= \emptyset && \text{(base case)} \\ B_b(D \cup \{e\}) &= \psi_b(B_b(D), e) && \text{(recursive step)} \end{aligned}$$

where $\psi_b(B_b(D), e)$ is the probabilistic function defined for all samples K of $B_b(D)$ as :

$$\begin{aligned} \Pr[\psi_b(K, e) = K] &= 1 - b \\ \Pr[\psi_b(K, e) = K \cup \{e\}] &= b \end{aligned}$$

As sketched in [27] and proved in the more general case of (ϵ, δ)-differential privacy [16], we benefit from improved privacy of $\sigma(\epsilon) = \ln(b.e^{\epsilon} + 1 - b)$.

Partitioning Property. The result is the same privacy cost as for parallel composition of a deterministic partitioning method.

Theorem 9. *Bernoulli partitioning has the probabilistic stability of* $\hat{\sigma}(\epsilon) = \epsilon$.

Proof. Instantiating Theorem 4 with ψ_b from the recursive definition, we reach the bound of ϵ as follows.

$$\begin{aligned} \hat{\sigma}(\epsilon) &= \ln(e^{(0+|0-1|)\epsilon}.\Pr[|\psi_b(K, e) \mathbin{\Delta} K| = 0] \\ &\quad + e^{(1+|1-1|)\epsilon}.\Pr[|\psi_b(K, e) \mathbin{\Delta} K| = 1]) \\ &= \ln e^{\epsilon}(b + (1-b)) = \epsilon \end{aligned}$$

It is reasonable not to expect a better result. Indeed, the privacy amplification of this sampling method occurs when b is small. Alas, in the context of partitioning, that means that the largest partition will be a sample of $1 - b$. As a consequence, whatever privacy amplification was achieved for the small half is lost on the big half.

5 Related work

Sampling and partitioning is explicitly used in several frameworks. Airavat [26] uses partitioning in its *MapReduce* framework. In the *sample and aggregation* framework used for computing *smooth sensitivity*, Nissim

et al. [19] uses random partitioning to divide the database into smaller databases. For simplification, instead of partitioning, the databases is constructed by taking random samples that are chosen independently of each other. When the number of clusters is low, each data item has the chance to appear in multiple samples. However the bounded variant of differential privacy is used (indistinguishability). As mentioned earlier, the privacy amplification effects of sampling are discussed in [27, 16]. In earlier work, Kasiviswanathan et al. [12] used this amplification effect to build private learners. Some works focus on data dependant partitioning of data to build histograms, Xiao et al. [30] research two partitioning strategies, cell-based and kd-tree based partitioning for building histograms and Kellaris and Papadopoulos [13] use a grouping technique based on sampling.

A non uniform sampling is used by Jorgensen et al.[11] which assigns a numerical value corresponding to the personal privacy preference for each database item. The personalised privacy then influences the inclusion probability of that individual in the sample used for differentially private analyses.

6 Conclusion

In this paper we have explored a variety of sampling methods and their effect on differential privacy in the strong case where the size of the dataset is private.

In particular we demonstrated a simple and practical attack on PINQ's deterministic sampling transformation which surprisingly shows that a sampling procedure may have a negative effect on the differential privacy cost of an analysis.

Further we investigated whether introducing randomness in selection of samples and partitions can significantly improve the differential privacy bounds compared to what their deterministic alternatives promise.

Technically, we have also shown how the concept of probabilistic stability provides a suitable generalisation of stability in systems that take advantage of composition principles to guarantee differential privacy.

While the choice of a sampling method is mainly driven by the environment and the application, we showed that, among common probability sampling and partitioning methods studied in this paper, (i) only Bernoulli sampling has the privacy amplification property and (ii) the corresponding partitioning algorithm fits perfectly with parallel query composition.

Further work includes the generalisation to approximate (ϵ, δ)-differential privacy and the implementation of the sampling mechanisms in a PINQ-style framework.

References

[1] Microsoft .NET Framework 4. Queryable.skip<tsource>method. `https://msdn.microsoft.com/en-us/library/bb357513%28v=vs.100%29.aspx`, 2016.

[2] Microsoft .NET Framework 4. Queryable.take<tsource>method. `https://msdn.microsoft.com/en-us/library/bb300906%28v=vs.100%29.aspx`, 2016.

[3] D. R. Bellhouse A. I. McLeod. A convenient algorithm for drawing a simple random sample. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 32(2):182–184, 1983. ISSN 00359254, 14679876.

[4] Ivan Rezucha C. T. Fan, Mervin E. Muller. Development of sampling plans by using sequential (item by item) selection techniques and digital computers. *J. of the Am. Stat. Assoc.*, 57(298):387–402, 1962. ISSN 01621459.

[5] Had00b Big data made simple. Reservoir sampling in mapreduce. `http://had00b.blogspot.se/2013/07/random-subset-in-mapreduce.html`, 2013.

[6] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. Calibrating noise to sensitivity in private data analysis. In *Proc. of the 3rd Conference on Th. of Crypt.*, TCC'06, pages 265–284, Berlin, Heidelberg, 2006. Springer-Verlag. ISBN 3-540-32731-2, 978-3-540-32731-8.

[7] Úlfar Erlingsson, Aleksandra Korolova, and Vasyl Pihur. RAPPOR: randomized aggregatable privacy-preserving ordinal response. *CoRR*, abs/1407.6981, 2014.

[8] Ronald Aylmer Sir Fisher and Frank Yates. *Statistical tables for biological, agricultural and medical research.* London : Oliver and Boyd, 1938.

[9] Marco Gaboardi, Andreas Haeberlen, Justin Hsu, Arjun Narayan, and Benjamin Pierce. Linear dependent types for differential privacy, 2013.

[10] Marco Gaboardi, Andreas Haeberlen, Justin Hsu, Arjun Narayan, and Benjamin C. Pierce. Linear dependent types for differential privacy. In *Proc. of the 40th Annual ACM SIGPLAN-SIGACT Symp. on Princ. of Prog. Lang.*, POPL '13, pages 357–370, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-1832-7.

[11] Z. Jorgensen, T. Yu, and G. Cormode. Conservative or liberal? personalized differential privacy. In *2015 IEEE 31st Int. Conf. on Data Eng.*, pages 1023–1034, April 2015.

[12] Shiva Prasad Kasiviswanathan, Homin K. Lee, Kobbi Nissim, Sofya Raskhodnikova, and Adam D. Smith. What can we learn privately? *SIAM J. Comput*, 40(3):793–826, 2011.

[13] Georgios Kellaris and Stavros Papadopoulos. Practical differential privacy via grouping and smoothing. In *Proceedings of the 39th in-*

ternational conference on Very Large Data Bases, PVLDB'13, pages 301–312. VLDB Endowment, 2013.

[14] Daniel Kifer and Ashwin Machanavajjhala. No free lunch in data privacy. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data*, SIGMOD '11, pages 193–204, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0661-4.

[15] Igor Kononenko and Matjaz Kukar. *Machine Learning and Data Mining: Introduction to Principles and Algorithms*. Horwood Publishing Limited, 2007. ISBN 1904275214, 9781904275213.

[16] Ninghui Li, Wahbeh Qardaji, and Dong Su. On sampling, anonymization, and differential privacy or, k-anonymization meets differential privacy. In *Proc. of the 7th ACM Symp. on Inf., Comp. and Comm. Sec.*, ASIACCS '12, pages 32–33, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1648-4.

[17] Frank McSherry. Privacy integrated queries. In *Proc. of the 2009 ACM SIGMOD Int. Conf. on Manag. of Data (SIGMOD)*. Association for Computing Machinery, Inc., June 2009.

[18] Prashanth Mohan, Abhradeep Thakurta, Elaine Shi, Dawn Song, and David Culler. Gupt: privacy preserving data analysis made easy. In *Proc. of the 2012 ACM SIGMOD Int. Conf. on Manag. of Data*, pages 349–360. ACM, 2012.

[19] Kobbi Nissim, Sofya Raskhodnikova, and Adam Smith. Smooth sensitivity and sampling in private data analysis. In *Proc. of the 39th Annual ACM Symp. on Th. of Comp.*, STOC '07, pages 75–84, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-631-8.

[20] Byung-Hoon Park, George Ostrouchov, Nagiza F Samatova, and Al Geist. Reservoir-based random sampling with replacement from data stream. In *SDM*, pages 492–496. SIAM, 2004.

[21] Byung-Hoon Park, George Ostrouchov, and Nagiza F. Samatova. Sampling streaming data with replacement. *Computational Statistics Data Analysis*, 52(2):750 – 762, 2007. ISSN 0167-9473.

[22] Davide Proserpio, Sharon Goldberg, and Frank McSherry. Calibrating data to sensitivity in private data analysis: A platform for differentially-private analysis of weighted datasets. *Proc. VLDB Endow.*, 7(8):637–648, April 2014. ISSN 2150-8097.

[23] Jason Reed and Benjamin C. Pierce. Distance makes the types grow stronger: A calculus for differential privacy. In *ACM SIGPLAN International Conference on Functional Programming (ICFP)*, 2010.

[24] Bengt Rosén. On sampling with probability proportional to size. *Journal of Statistical Planning and Inference*, 62(2):159 – 191, 1997. ISSN 0378-3758.

[25] Bengt Rosén. Asymptotic theory for order sampling. *Journal of Statistical Planning and Inference*, 62(2):135 – 158, 1997. ISSN 0378-3758.

[26] Indrajit Roy, Srinath T. V. Setty, Ann Kilzer, Vitaly Shmatikov, and Emmett Witchel. Airavat: Security and privacy for mapreduce. In *Proc. of the 7th USENIX Conf. on Networked Sys. Design and Impl.*, NSDI'10, pages 20–20, Berkeley, CA, USA, 2010. USENIX Association.

[27] Adam Smith. Differential privacy and the secrecy of the sample. `https://adamdsmith.wordpress.com/2009/09/02/sample-secrecy/`, 2009.

[28] Y. Tillé. *Sampling Algorithms*. Springer Series in Statistics. Springer, 2006. ISBN 9780387308142.

[29] Jeffrey S. Vitter. Random sampling with a reservoir. *ACM Trans. Math. Softw.*, 11(1):37–57, March 1985. ISSN 0098-3500.

[30] Yonghui Xiao, Li Xiong, and Chun Yuan. *Differentially Private Data Release through Multidimensional Partitioning*, pages 150–168. Springer Berlin Heidelberg, 2010. ISBN 978-3-642-15546-8.

Appendix

1.1 Proof of Concept to Demonstrate the PINQ's Weakness

The programming frameworks introduced in this paper comes with the guarantee that an analyst with the permission to run arbitrary analyses on a database will not learn more than the limited private budget specified by the administrator. An attack that can break any of the following guarantees, beyond the limited probability of e^{ϵ}, is considered a successful privacy breach:

- Determine whether an individual is in the database or not.
- Extract the individuals properties (age, salary, ...).

Given information about an item i (here number 7), the goal of proof of concept code in Listing 1 (simplified in Algorithm 3) is to determine the presence of the item in the dataset D. It is easy to show that the second attack is a form of the first attack if the attacker enumerate over possible value of the property to find which value in the domain of the private property appears in the dataset.

The first step in the proof of concept is to partition the dataset D into two parts according to satisfying the equality with the target element i. As a result $K = \{x | x \in D, \lambda x \to x = i\}$ and $\bar{K} = D \setminus K$. Note that K and $\bar{K}$ share ϵ budget since they are disjoint subsets and PINQ's budget management system request only ϵ when two ϵ-differential private query are executed on K and $\bar{K}$. Figure 1 demonstrate how request for privacy budget flows between intermediate protected objects of Algorithm 3 when a query with ϵ cost is executed on the protected object containing the result of $\bar{K} \cup \text{Take}(K \cup r, 1)$.

The following transformations that are used in this sample code don't scale up the stability since they are all 1-stable over their arguments:

– addition of r to the K using union
– taking the first element
– merging the result with the partition $\bar{K}$ using union

If item i is present in D, the partition K has only the item i. In this case taking the first element of $K \cup r$ gives i and when we merge $\bar{K}$ and Take$(K \cup r, 1)$ together using union transformation, we end up with the starting dataset D. The interesting case happens when i is not in D, in this case K is empty and take returns the random item r. The final merge (union) of $\bar{K}$ and Take$(K \cup r, 1)$ results in $D \cup r$. Performing this procedure several times to add several arbitrary random items makes the outcomes so different that they are distinguishable even after a differentially private counting query with a low value of ϵ.

Algorithm 3 Proof of concept code that reveals the existence of the item i in the protected dataset D

1: **procedure** REVEAL(i, D)
2: **for** $s \leftarrow 1, scale$ **do**
3: $(K, \bar{K}) \leftarrow$ **Partition**$(D, \lambda x \rightarrow (x == i))$
4: $r \leftarrow$ **RandomItem**() ▷ random r $(r \neq i)$
5: $D \leftarrow$ **Take**$(K \cup r, 1) \cup \bar{K}$
6: **return NoisyCount**$_\epsilon(D)$

Listing 1. Proof of concept code

```
// Secure initialisation
double[] rawData =
  new double[] {1,2,3,4,5,6,8,9,10};
var sourceD = rawData.ToArray().AsQueryable();
var agent= new PINQAgentBudget(1.0);
var D = new
  PINQueryable<double> (sourceD, agent);

double[] invalid = new double[] { 999 };
double i = 7; // The item we are looking for
int scale = 1000;
bool[] keys = new bool[] {true,false} ;
for (int s = 0; s < scale; s++) {
  invalid[0] += 0.0001; // a unique item (random)
  var r = invalid.ToArray().AsQueryable();
  var parts = D.Partition
    (keys, x => (x==i)? true :false);
  var K = parts [true];
  var K_ = parts [false];
  D = K_.Union(
    K.Union(r).Take (1)
```

```
  );
}
Console.WriteLine("Noisy result:\t\t{0:F4}",
  D.NoisyCount(0.01));
```

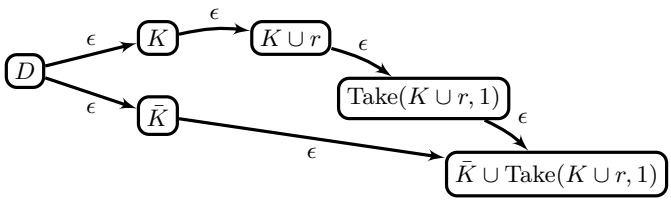

Fig. 1. privacy budget request between protected objects

1.2 Probabilistic Stability of Uniform Sampling (Without Replacement)

Proof. We want to find the probabilistic stability $\sigma(\epsilon)$ such that $e^{-\sigma(\epsilon)} \leq \frac{\Pr[M(RS(D\cup\{e\}))\subseteq S]}{\Pr[M(RS(D))\subseteq S]} \leq e^{\sigma(\epsilon)}$. We fix S, an arbitrary query result in the range of M. By using the recursive form of $RS(D \cup \{e\}))$ (Theorem 5) and the law of total probability over $\overrightarrow{RS(D)}$, we have:

$$\begin{aligned}
&\frac{\Pr[M(RS(D \cup \{e\})) \subseteq S]}{\Pr[M(RS(D)) \subseteq S]} \\
&= \frac{\Pr[M(\psi_D(RS(D), e)) \subseteq S]}{\Pr[M(RS(D)) \subseteq S]} \\
&= \frac{\sum_{K \in \overrightarrow{RS(D)}} \Pr[M(\psi_D(K, e)) \subseteq S].\Pr[RS(D) = K]}{\sum_{K \in \overrightarrow{RS(D)}} \Pr[M(K) \subseteq S].\Pr[RS(D) = K]}
\end{aligned}$$

Then, by equality of all samples probability (Definition 6), we derive:

$$= \frac{\alpha.\sum_{K \in \overrightarrow{RS(D)}} \Pr[M(\psi_D(K, e)) \subseteq S]}{\alpha.\sum_{K \in \overrightarrow{RS(D)}} \Pr[M(K) \subseteq S]}$$

Let us define $K^{\pm i} = \{K' \big| |K \,\Delta\, K'| = i\}$ a function defining Hamming distance-based equivalence classes on neighbourhoods. Thus, by law of total probability over $\mathbb{N}$, we have:

$$= \frac{\sum_{K \in \overrightarrow{RS(D)}} \sum_{i \in \mathbb{N}} \Pr[\psi_D(K, e) \in K^{\pm i}].\Pr[M(K^{\pm i}) \subseteq S]}{\sum_{K \in \overrightarrow{RS(D)}} \Pr[M(K) \subseteq S]}$$

We now give an over-approximation by relying on the fact that all probabilities are greater than zero. By differential privacy (Definition 1), we have $\Pr[M(K^{\pm i}) \subseteq S] \leq e^{i\epsilon} \Pr[M(K) \subseteq S]$. Similarly for the lower-bound, we have
$e^{-i\epsilon} \Pr[M(K) \subseteq S] \leq \Pr[M(K^{\pm i}) \subseteq S]$.

Thus, for any i,

$$\leq \frac{\sum_{i\in\mathbb{N}} \sum_{K\in\overrightarrow{RS(D)}} \Pr[M(K)\subseteq S].e^{i\epsilon}.\beta_i}{\sum_{K\in\overrightarrow{RS(D)}} \Pr[M(K)\subseteq S]}$$

with $\beta_i = \Pr[\psi_D(K,e)\in K^{\pm i}]$.

Finally, by Observation 1 in [7] and simplification, we get the following upper-bound:

$$\leq \sum_{i\in\mathbb{N}} e^{i\epsilon}.\Pr[|\psi_D(K,e)\,\Delta\,K| = i]$$

We have a similar lower-bound by the same approximation.

This allows us to conclude the following result:
$\sigma(\epsilon) \leq \ln \sum_{i\in\mathbb{N}} e^{i\epsilon}.\Pr[|\psi_D(K,e)\,\Delta\,K| = i]$ for any K, D, e, and ψ.

1.3 Probabilistic Stability of Uniform Partitioning (Without Replacement)

Proof. We follow a similar proof technique and almost identical steps as the previous part, to find probabilistic stability $\hat{\sigma}(\epsilon)$ such that $e^{-\hat{\sigma}(\epsilon)} \leq \frac{\Pr[\hat{M}(\hat{RS}(D\cup\{e\}))\subseteq S]}{\Pr[\hat{M}(\hat{RS}(D))\subseteq S]} \leq e^{\hat{\sigma}(\epsilon)}$ we fix $S = \langle S_1, S_2\rangle$ to be a set of pairs of results in the range of $\hat{M}$ [1]. Using recursive form of RS and law of total probability.

$$\frac{\Pr[\hat{M}(\hat{RS}(D\cup\{e\}))\subseteq S]}{\Pr[\hat{M}(\hat{RS}(D))\subseteq S]} = \frac{\sum_{\langle K,\overline{K}\rangle\in\overrightarrow{\hat{RS}(D)}} \left[\begin{array}{l}\Pr[\hat{RS}(D)=\langle K,\overline{K}\rangle].\\ \quad \Pr[\hat{M}(\hat{\psi_D}(\langle K,\overline{K}\rangle,e)\subseteq S]\end{array}\right]}{\sum_{\langle K,\overline{K}\rangle\in\overrightarrow{\hat{RS}(D)}} \left[\begin{array}{l}\Pr[\hat{RS}(D)=\langle K,\overline{K}\rangle].\\ \quad \Pr[\hat{M}(\langle K,\overline{K}\rangle)\subseteq S]\end{array}\right]}$$

Knowing $K\cup\bar{K} = D$, after introducing e, $K'\cup\bar{K'} = D\cup\{e\}$ such that K' is the result of function $\psi_D(K,e)$, if $|K\,\Delta\,K'| = i$ then $|\bar{K}\,\Delta\,\bar{K'}| = |i-1|$.

$$\frac{\alpha.\sum_{K\in\overrightarrow{RS(D)}} \left[\sum_{i\in\mathbb{N}} \left[\begin{array}{l}\Pr[\hat{\psi_D}(\langle K,\overline{K}\rangle,e)\in\langle K^{\pm i},\overline{K^{\pm i}}\rangle].\\ \quad \Pr[\langle M_1(K^{\pm i}),M_2(\overline{K^{\pm i}})\rangle\subseteq S]\end{array}\right]\right]}{\alpha.\sum_{K\in\overrightarrow{RS(D)}} \Pr[\langle M_1(K),M_2(\overline{K})\rangle\subseteq S]}$$

[1] $\hat{RS}$ is lifted function from sampling method RS that creates two disjoint partitions K and $\overline{K}$. Similarly $\hat{M}$ is lifted function that run two queries M_1 and M_2 on these two partitions.

$$= \frac{\sum\limits_{K \in \overrightarrow{RS(D)}} \sum_{i \in \mathbb{N}} \beta_i . \Pr[\langle M_1(K^{\pm i}), M_2(\overline{K^{\pm i}})\rangle \subseteq S]}{\sum\limits_{K \in \overrightarrow{RS(D)}} \Pr[\langle M_1(K), M_2(\overline{K})\rangle \subseteq S]}$$

$$\leq \frac{\sum_{i \in \mathbb{N}} \sum\limits_{K \in \overrightarrow{RS(D)}} \beta_i . \begin{bmatrix} e^{i\epsilon} . \Pr[M_1(K) \subseteq S_1] \\ e^{|i-1|\epsilon} . \Pr[M_2(\overline{K}) \subseteq S_2] \end{bmatrix}}{\sum\limits_{K \in \overrightarrow{RS(D)}} \Pr[M_1(K) \subseteq S_1] . \Pr[M_2(\overline{K}) \subseteq S_2]}$$

with $\beta_i = \Pr[\psi_D(K, e) \in K^{\pm i}]$.

Simplifying the expression we have probabilistic stability of $\hat{\sigma}(\epsilon) = \ln \sum_{i \in \mathbb{N}} e^{(i+|i-1|)\epsilon} . \Pr[|\psi_D(K, e) \, \Delta \, K| = i]$

1.4 Recursive (Fixed Sized) Uniform Sampling (Without Replacement)

Proof. In the case where $n \geq |D \cup \{e\}|$, the probability of the sample to be $D \cup \{e\}$ is 1.

For the case where $n < |D \cup \{e\}|$, we rely on the inductive definition of $R_n(D \cup \{e\})$. Two cases are possible:

$$\begin{aligned}
&\textbf{Case } e \in X : \Pr[R_n(D \cup \{e\}) = X] \\
&= \sum_{a \in D \cup \{e\} \setminus X} \begin{bmatrix} \Pr[R_n(D) = X \cup \{a\} \setminus \{e\}] \\ \Pr[\psi_{n,|D|}(X \cup \{a\} \setminus \{e\}, e) = X] \end{bmatrix} \\
&= \underbrace{(|D| + 1 - n)}_{\text{from summation}} . \binom{|D|}{n}^{-1} . (|D| + 1)^{-1} \\
&= \binom{|D| + 1}{n}^{-1} \\
&\textbf{Case } e \notin X : \Pr[R_n(D \cup \{e\}) = X] \\
&= \Pr[R_n(D) = X] . \Pr[\psi_{n,|D|}(X, e) = X] \\
&= \binom{|D|}{n}^{-1} . \frac{|D| + 1 - n}{|D| + 1} \\
&= \binom{|D| + 1}{n}^{-1}
\end{aligned}$$

In both cases, $|D| + 1 - n$ makes match the expected probabilities.

Paper IV : PreTPost: A Transparent, User Verifiable, Local Differential Privacy Framework

Under Submission

PreTPost: A Transparent, User Verifiable, Local Differential Privacy Framework

Hamid Ebadi and David Sands

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

Abstract. Differential privacy assures the privacy of individuals by not revealing too much about them when their data is used in an aggregated statistical analysis. A differentially private data analysis usually works by the controlled addition of noise to the computation, aiming for enough to achieve a given degree of privacy at the same time as still giving statistically useful results. Most methods and frameworks supporting differentially private analyses are based on the assumption of a centralised sensitive database managed by a trusted curator, which is not the ideal model in many practical scenarios. In practical applications such as Google Chrome's collection of statistics on browsing history, or Apple's training of predictive messaging in iOS 10, data is not collected before analysis, but uses differential privacy locally at the source.

In this paper we introduce a general framework targeting this local setting in which there is no centralized database or trusted curator, and the differential privacy mechanism must be applied at the data source. The framework is general in the sense that it allows the user to receive new analyses on the fly. At the same time, it supports locally verifiable privacy: with just a simple calculation the user is able to determine the privacy cost of the incoming analysis request, meaning that, unlike Google or Apple's use of local differential privacy, we do not have to trust the analyst. Combining this with a personal privacy budget (aka personalised differential privacy), gives the user the ability to express a variety of personal privacy policies with differential privacy as the baseline guarantee.

The method leverages the simple but new observation that local differential privacy is preserved by arbitrary data pre-processing. The PreTPost framework requires a data analysis to be decomposed into a pre-processor (Pre), a simple core probabilistic transformation (T), and a post-processor (Post), where a simple analysis of T is sufficient to calculate the differential privacy cost. We have verified that this decomposition is possible and straightforward for a range of local analysis algorithms from the literature.

We also outline a prototype implementation of the PreTPost framework using system-level sandboxed execution.

1 Introduction

It is possible to perform statistically useful analyses on a collection of user data without incurring any significant privacy risk to any one individual, by accepting a certain degree of randomness or noise in the outcome of the analysis. Differential privacy [11] is a rigorous mathematical definition of privacy which supports this approach. Using differential privacy as a foundation, it has been shown that a number of statistical analyses can be computed in a differentially private way.

As an example, suppose that we wish to answer the question: *what percentage of browser users have downloaded copyrighted material from the internet?* Suppose this information can be determined from the browsing history stored in your browser. One way to give a useful but necessarily imprecise answer to this question, at the same time as bounding privacy risk for the individual, is to use the following procedure: each user flips two coins; if they are different then answer the query truthfully, if they are both heads answer "Yes", and "No" if they are both tails. The "aggregator" can now make a statistical estimate of the true percentage: if there are y "Yes" answers from 1000 respondents then we expect 250 random "Yes" answers, 250 random "No" answers, so the fraction of downloaders can be estimated as $(y-250)/500$. At the same time, anyone intercepting a "Yes" answer from any one individual cannot know whether it was generated by honesty or randomness. In this specific example the increase in risk to the individual is a multiplicative factor of $0.75/0.25 = 3$, so if there was already a 0.1% chance of someone discovering that a particular individual was downloading copyrighted material without the user participating in the survey, the risk in participation would at most increase to $3 \times 0.1\%$ if the user participated. By adjusting the probabilities of the coin flips one can lower this risk factor at the expense of lower accuracy in the reported result.

This algorithm is an example of a mechanism which satisfies the definition of differential privacy [11] (it is ln(3)-differentially private, to be precise, where ln is the natural logarithm). The controlled addition of noise (and hence an imprecise answer), is a necessary feature of any differentially private mechanism. The particular method described in the example, an anonymising survey technique called *randomised response*, actually pre-dates the definition of differential privacy by some 50 years [45].

Frameworks for Differential Privacy Differential privacy is (only) a precise mathematical definition of privacy for a query mechanism – we postpone giving the formal definition until the next section – but does not

in itself prescribe how to compute differentially private approximations to interesting queries. A number of differential privacy frameworks have emerged which support the construction of differentially private mechanisms. These leverage general compositional properties of differential privacy to simplify the static verification or dynamic enforcement of a desired amount of privacy. Examples of systems of this ilk are, for example,

- PINQ [30], wPINQ [36], and ProPer [15], which dynamically monitor how data is used, and ensure that the computation never exceeds a given privacy budget, or
- Fuzz [20, 18] where programs are written in a functional language with a custom type system for which static type checking provides differential privacy guarantees.
- Offline verification methods and tools which employ interactive theorem proving [6] tailored to prove the differential privacy of imperative algorithms.

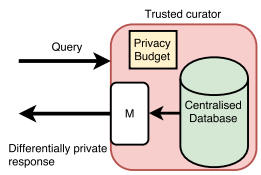

Fig. 1: Centralised Model

All of these frameworks focus on verification of mechanisms which are assumed to have access to the whole data set. This implies the existence of a trusted database curator who holds the sensitive data, and who has the responsibility to apply the mechanism to the data and to keep track of a global privacy risk (Figure 1).

Local Differential Privacy The randomised response mechanism described in the example above has a different trust model, one which constrains the way differential privacy can be implemented, but which has many potential benefits. This is sometimes called *local differential privacy* [9, 28] or simply "the local model" [12]. The constraint is that the privacy mechanism is applied locally, at each participant. The benefits accrue from the fact that there is no longer a need to centrally store sensitive data – privacy is managed at the source (your cell-phone, car, web browser...). In practice this could have significant benefits regarding security since it eliminates the single point of failure, and for legal aspects of data storage, since it could be argued that sensitive data is not centrally stored at all.

Additionally it lowers the consequences of data breach (external attackers) and data leaks (by insiders), and finally, it is compatible with several attack models in which the attacker *partially* breaches the system or the communication channel. There are even potential efficiency benefits from distributing the computation.

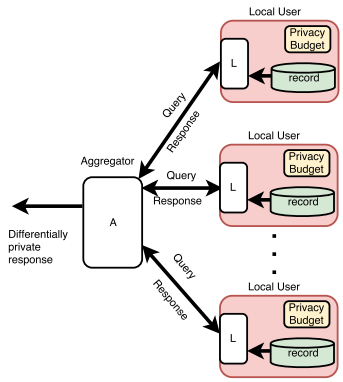

Fig. 2: Local Model

Perhaps for these reasons, the local model is the flavour of differential privacy which has been implemented in the actual "real-world" instances of differential privacy, by Google (in the Chrome browser) [16] [17], and Apple (in iOS 10 and MacOS) [5].

The analyses by Apple and Google still require a great deal of trust – you have to trust that they implemented their algorithms correctly on your device, and that the algorithms are indeed differentially private, not just for a single round of communication, but even when statistics are reported over time. Apple, in particular, keep both the algorithms and the intended quantity of privacy ("epsilon") secret, and a recent study of their algorithms by Tang et al.[40] suggests that this trust is not well founded, and concludes

> "We call for Apple to make its implementation of privacy-preserving algorithms public and to make the rate of privacy loss fully transparent and tuneable by the user"

What we desire, and what provide in this work, is a framework for local differential privacy in which has the following features:

Open Design The framework should be open-ended, allowing the analysts to design new queries that are not hard-wired into the system.

Verifiable Privacy The framework should provide verifiable privacy for the user; on receiving a new query algorithm, the user should be

able to independently verify its privacy cost. Differentially private systems are probabilistic which complicates the black-box testing of their correctness. In addition, not only the result of the algorithm but also the procedure has to be checked for side channels (e.g. timing side channel). As an example some vulnerabilities have been found in the privacy guarantee of existing local algorithms [10] as shown in [33].

Privacy Control The all-or-nothing approach enforced by existing systems is usually not welcomed by users. Giving control over a privacy budget and empowering users to adjust the desired level of privacy may encourage more individuals to contribute.

Contributions We propose the first framework for verified local differential privacy that allows a data analyst, curious and possibly untrusted, to collect and perform statistical analysis on sensitive personal data while providing the users with a verifiable differential privacy guarantee without users needing to trust the analyst or the mechanism which is transmitted. Our approach is built on (i) simple properties of local differential privacy mechanisms – the observation that they are preserved by both pre-processing and post-processing of data (Section 2), together with (ii) the idea of *personalised differential privacy* [26, 15] which allows users to manage their own privacy budget locally. Personalised differential privacy offers other benefits, such as:

Availability Not all agents (users) need to participate in all analyses at the same time, so analysts can sample freely without fear of wasting a global budget or keeping track of partitions of the user-space.

Dynamic data Unlike systems with a global privacy budget, the personalised method is applicable when individuals are joining or leaving the system continuously.

Distribution Instead of tracking budgets for each user, we can outsource the task to each user.

Our framework proposed a simple generic structure for local differential privacy algorithms which permits the privacy cost to be verified by a simple arithmetic calculation [28, 44] over the randomisation at the heart of the mechanism. We provide a prototype implementation of the framework (Section 3.4) which employs client-side sandboxing to control access to the private data and to mitigate simple timing channels. The approach is evaluated (Section 4) by showing how a range of locally differentially private algorithms, including algorithms from Google's *RAPPOR* system [16, 17] can be implemented in the framework.

2 Foundations

In this section we give the basic definitions of (local) differential privacy, and the properties that we build on.

2.1 Differential Privacy

The definition of differential privacy we start with assumes that there is data from a known collection of n users, represented by a vector of values in some domain V. An analysis is represented here by a randomised function.

Definition 1 (Differential Privacy). *A randomised function M is ε-differentially private, where ε is a non-negative real number, if for all $n > 0$ and all input vectors $\boldsymbol{u}, \boldsymbol{v} \in V^n$ such that $\boldsymbol{u}$ and $\boldsymbol{v}$ differ at just one index position, then for all $S \subset range(M)$*

$$\Pr[M(u) \in S] \leq \exp(\varepsilon) \times \Pr[M(v) \in S]$$

This variant of differential privacy is sometimes referred to as *bounded* differential privacy[13, 29], because it does not seek to protect the size of the collection (the presence or absence of individuals), only the contents.

We define local differential privacy in a slightly nonstandard way, by viewing it as regular differential privacy with additional constraints. (The standard definition will then just be a consequence of our definition). A local differentially private algorithm is a way to achieve differential privacy by structuring the mechanism M to use local processing L on each of the n input data sources before sending them to an aggregator A which combines them into the desired result. See Figure 2 (ignoring the budget component for now).

We furthermore allow the local mechanism L to have access not only to the local private data, but also some local but non-private data $p \in P$. This latter component may be useful in algorithms that involve multiple rounds, where the non-private local data may include information obtained from the server. We will also use it shortly to model a personal privacy budget.

We put these together we get a (somewhat redundant) definition of local differential privacy:

Definition 2 (Local Differential Privacy). *Suppose P is the domain of local public information and V is the domain of private data. Let L be a randomised function $P \times V \to W$ for some finite domain W, and A be a (possibly randomised) function $(\bigcup_{n>0} W^n) \to X$.*

We define the pair $\langle L, A \rangle$ to be ε-locally differentially private if

1. *For all $p_1, \ldots, p_n \in P$, the function $M(\boldsymbol{x}) = A(L(p_1, x_1), \ldots L(p_n, x_n))$ is ε-differentially private, and*

2. *The function* $M'(\boldsymbol{x}) = (L(p_1, x_1), \ldots, L(p_n, x_n))$ *is also* ε*-differentially private.*

The redundancy in this definition can be expressed as follows:

Proposition 1. *Local mechanism* $\langle L, A\rangle$ *is* ε*-locally differentially private if and only if for all* $p \in P$ *and all* $u, v \in V$*, and* $w \in W$*,*

$$\Pr[L(p, u) = w] \leq \exp(\varepsilon) \times \Pr[L(p, v) = w]$$

This follows from the facts that the requirement (2) in the definition subsumes requirement (1) using the fact that differential privacy is closed under post-processing [12]. The finiteness of W is sufficient to allow us to test the ε-distance property at each value w. In many works [9, 28, 7, 23] Proposition 1 is given as the *definition* of local differential privacy.

In what follows we will ignore the aggregator A, since it plays no role in the privacy of the mechanism (it does, of course, play a role in understanding the utility with respect to the ideal non-private computation, but utility analysis is outside the scope of this paper). In what follows we will view local differential privacy as a local mechanism L satisfying the property described in Proposition 1.

2.2 Composition Principles

Frameworks for differential privacy all build on composition principles – ways of deducing the differential privacy of algorithms from the differential privacy and other properties of their components. For local differential privacy there are two principles that mirror standard principles of regular differential privacy, namely the simple composition of differentially private mechanisms, and the robustness of results under post processing. The proofs are essentially the same as in the case of regular differential privacy (see e.g. [30]).

Proposition 2 (Simple Sequential Composition). *The composite local differentially private mechanism which first applies* L_1 *(an* ε_1 *local differential privacy algorithm) then* L_2 *(an* ε_2 *local differential privacy algorithm), returning the pair of results, has* $\varepsilon_1 + \varepsilon_2$ *local differential privacy.*

Note here that the choice of L_2 and its privacy cost may depend adaptively on the value delivered by L_1.

Theorem 1 (Post Processing). *Suppose* L *has* ε *local differential privacy, and that* M *is a probabilistic function over the codomain of* L*. Then the function* $M \circ L$ *(i.e.* M *is applied to the result of* L*), has* ε *local differential privacy.*

Here the operation M is a post-processor for L. Post processing cannot make the result any more sensitive than it already is – although it could of course make the result even less sensitive, since like standard differential privacy, if L has ε local differential privacy then it also has ε' local differential privacy for any $\varepsilon' > \varepsilon$. Post processing argument in the centralised setting is discussed e.g. in Proposition 2.1 by Dwork and Roth [12]. The proof here is equally straightforward.

We now come to a simple but key contribution of this work, a *pre-processing* theorem. The idea here is to characterise how pre-processing of the private data influences local differential privacy. Pre-processing theorems play a key role in systems like PINQ or Fuzz, where data pre-processing scales the privacy cost according to the *sensitivity* of the pre-processing operation. We won't go into details of sensitivity analysis in the central model since pre-processing for local differential privacy is considerably simpler:

Theorem 2 (Pre Processing). *Suppose M has ε local differential privacy, where $V = \text{domain}(M)$ is finite, and that pre is a probabilistic function producing values in the domain of M. Then the function $M \circ pre$ (i.e. pre is pre-processor to M), has ε local differential privacy.*

Proof. Assume M such that $\forall_{v,u \in V}.Pr[M(v) = w] \leq Pr[M(u) = w] \cdot e^{\varepsilon}$ and let v_x in the finite V such that it minimizes $Pr[M(.) = w]$. $(\forall_{v_i \in V}.Pr[M(v_x) = w] \leq Pr[M(v_i) = w])$. We show for any randomized mapping $pre : D \to V$ we have $\forall_{d,c \in D} Pr[M(pre(d)) = w] \leq Pr[M(pre(c)) = w] \cdot e^{\varepsilon}$

$$
\begin{aligned}
& Pr[M(pre(d)) = w] && \text{(left hand side)} \\
&= \sum_{v_i \in V} Pr[M(v_i) = w] \cdot Pr[pre(d) = v_i] \\
&\leq \sum_{v_i \in V} Pr[M(v_x) = w] \cdot e^{\varepsilon} \cdot Pr[pre(d) = v_i] && \text{(by the first assumption)} \\
&= Pr[M(v_x) = w] \cdot e^{\varepsilon} \cdot \sum_{v_i \in V} Pr[pre(d) = v_i] \\
&= Pr[M(v_x) = w] \cdot e^{\varepsilon} \cdot \sum_{v_i \in V} Pr[pre(c) = v_i] && \text{(since sums are both one)} \\
&= e^{\varepsilon} \cdot \sum_{v_i \in V} Pr[M(v_x) = w] \cdot Pr[pre(c) = v_i] \\
& && \text{(by the second assumption)} \\
&\leq e^{\varepsilon} \cdot \sum_{v_i \in V} Pr[M(v_i) = w] \cdot Pr[pre(c) = v_i] \\
&= e^{\varepsilon} \cdot Pr[M(pre(c)) = w]
\end{aligned}
$$

Note that the domain of *pre* need not be finite or indeed discrete. For example the preprocessor might take the GPS coordinates of the user's vehicle and its speed, and return a bit denoting whether the vehicle is speeding or not.

These composition properties form core of our framework – they embody the simple observation that we can freely pre-process data, and post-process it, without adversely influencing the privacy guarantees of a differentially private mechanism.

2.3 Randomised Response

In order to fruitfully apply the pre- and post-processing theorems we must have some starting point – some "atomic" differentially private core algorithms. In prior frameworks (*ibid.*) this is usually achieved by reference to known differentially private methods (such as adding Laplace noise scaled according to the sensitivity of a known function). In the context of local differential privacy, the core (with one or two exceptions discussed in Section 3.3) is a "randomised response" mechanism expressed as a simple stochastic matrix. The differential privacy cost of such a matrix can be calculated *in situ*.

Randomized response introduced by Warner in 1965 [45] uses data(input) perturbation[1] [1] for statistical databases. The coin-flipping example given in the introduction is one way of implementing such a scheme. We represent such an algorithm as a *right stochastic matrix*, [2]. In the case of the example in the introduction the stochastic matrix is the one given below:

$$\begin{array}{c} \\ \text{Yes} \\ \text{No} \end{array} \begin{array}{c} \begin{array}{cc} \text{Yes} & \text{No} \end{array} \\ \begin{pmatrix} 0.75 & 0.25 \\ 0.25 & 0.75 \end{pmatrix} \end{array} \tag{1}$$

We assume there exists a canonical enumeration (written $\overline{V} = (v_1, \ldots, v_{|V|})$ of the elements of any finite domain V. In this case $\overline{V} = (\text{Yes}, \text{No})$. 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, then, that each row is a probability distribution (non-negative reals which sum to 1).

We interpret a stochastic matrix T as a probabilistic function, writing $T(v)$ for the action of T on value v, where $\Pr[T(v_i) = v_j] = T_{ij}$. For the sake of simplicity we will assume that the input and output domains are the same (although this is not an essential feature since we will not need to iterate the application of a matrix).

[1] other methods are conceptual, query restriction, output perturbation

[2] also known as probability matrix, transition matrix, substitution matrix, and Markov matrix

Definition 3. *Let T be a stochastic matrix. Define $cost(T) \in \mathbb{R}^{\geq 0} \cup \{\infty\}$ as*

$$cost(T) = \ln\left(\max_j\left(\frac{\max_i T_{i,j}}{\min_{i'} T_{i',j}}\right)\right)$$

where for the purposes of this definition we define $x/0 = \infty$.

The cost is the natural log of the largest ratio between the largest and smallest probability of any particular result. For example in the matrix 1, the value for the privacy cost of the matrix is $\ln(0.75/0.25)$.

Theorem 3. *Stochastic matrix T has $cost(T)$-local differential privacy.*

Here ∞-local differential privacy is the trivial property satisfied by all functions.

Proof. Take any two values v_m and v_n (corresponding to rows m and n of the matrix respectively). We need to show that for all j, $\Pr[T(v_m) = v_j] \leq e^{cost(T)}.\Pr[T(v_n) = v_j]$, which is equivalent to $\forall j.T_{m,j} \leq e^{cost(T)}.T_{n,j}$. If $T_{m,j}$ and $T_{n,j}$ are both zero then we are done; when one is zero and the other is not then $cost(T) = \infty$ and the statement is vacuous. Otherwise we require that $T_{m,j}/T_{n,j} \leq e^{cost(T)}$, and this follows easily from the definition of *cost*, which is the natural log of the maximum possible value reached by $T_{m,j}/T_{n,j}$.

2.4 Personalised Privacy

Our framework makes it easy for a user to verify local differential privacy by delivering the query in the form of a pre-processor, a stochastic matrix, and a post-processor: the cost is just the cost of the matrix. The simple composition property means that we can keep track of the privacy cost by adding the cost of queries over time. But privacy cost (the epsilon) is not just something that we may want to measure, it is something that we usually want to *enforce*. If the privacy budget is global (the global differential privacy goal) then how can it be enforced locally?

Our solution here is to move to a generalisation to differential privacy called *personalised differential privacy* (PDP). PDP, with the same name and same definition, was proposed at around the same time in two independent works: [15, 26] The idea is that each participant has there own personal privacy budget. In [26] it is motivated that each user should decide on their own risk profile and set their budget accordingly, whereas in [15] the purpose is to provide a more fine-grained account of privacy cost, but it is assumed that all users start out with the same budget.

It should be noted at this point that the combination of local and personalised differential privacy has been studied in a third paper [8]. However these works add an additional feature, a weakening of differential privacy, that we will not use here.

We take the definition of PDP as formulated in [15] and adapt it to our setting, vectors of input values of known length:

Definition 4 (Personalised (Big Epsilon) Differential Privacy). *Let $\mathscr{E}$ be a function from $\mathbb{Z}^+$ to non-negative real numbers. A randomized query Q provides $\mathscr{E}$-*differential privacy *if for all vectors $\boldsymbol{u}, \boldsymbol{v} \in V^n$ such that $\boldsymbol{u}$ and $\boldsymbol{v}$ differ at just index i, then for all $S \subset range(M)$*

$$\Pr[M(\boldsymbol{u}) \in S] \leq \exp(\mathscr{E}(i)) \times \Pr[M(\boldsymbol{v}) \in S]$$

Personalised differential privacy can be viewed as a way to allow users to determine their own privacy levels [26]. But alternatively [15], it can be seen as an alternative flexible way to enforce regular differential privacy under circumstances where users might arrive or leave the system over time, or where not all queries are relevant to all users. This latter view in our opinion fits our perspective well: sometimes the aggregator may choose a particular subset of users to pose a survey, and sometimes users might not participate simply because they are not accessible. A personalised budget smoothly handles these situations, as well as our desire to have locally verifiable privacy.

The definition of personalised local differential privacy is just a slight generalisation of the existing notion where we assume that the local data p_i always includes the index i (i.e. there is a function *index* with an invariant $index(p_i) = i$. Assuming that this is invariant, then we can define local personalised differential privacy

Definition 5. *Local mechanism L is $\mathscr{E}$-locally differentially private if and only if for all $p \in P$ and all $u, v \in V$, and $w \in W$,*

$$\Pr[L(p, u) = w] \leq \exp(\varepsilon) \times \Pr[L(p, v) = w]$$

where $\varepsilon = \mathscr{E}(index(p))$.

It follows that for all $p_1, \ldots, p_n$ such that $index(p_i) = i$ the function $M'(\boldsymbol{x}) = (L(p_1, x_1), \ldots, L(p_n, x_n))$ is also $\mathscr{E}$-differentially private, which in turn implies that it is ε-differentially private with $\varepsilon = \sup_{i \in \mathbb{N}} \mathscr{E}(i)$.

3 Framework

The previous section covered the foundations for our framework. In this section we outline how the components are put together.

The framework prescribes the interaction between two parties, the *user* (the user who contributes the data) and the *data aggregator* (the analyst). There are many users interacting with a single aggregator (a requirement for differential privacy to be useful).

We take advantage of the pre- and post-processing properties of local differential privacy to give a canonical representation of a local differential

private mechanism which is transmitted from the aggregator to the users, namely a quadruple

$$\langle t, pre, T, post \rangle$$

where *pre* is the *pre-processor*, a data pre-processing function, T is a randomised response matrix, and *post* is a *post-processor* for the output of T. The values t is an estimate for the maximum time expected for executing the mechanism, and is used to prevent covert timing channels (cf. [20]).

The rough idea is that the user will (i) determine the privacy cost by inspecting $cost(T)$, and if this is favourable, (ii) execute the function $post \circ T \circ pre$ on the sensitive data to produce the intended output.

The details are a little more involved on the user side, since the framework accommodates both the local state, which would typically include a privacy budget, and two local policy functions, a public one which determines how and when the budget can be consumed, and a private one which may make further decisions on the use of data in a potentially sensitive way.

In what follows we will detail the expectations on the aggregator side, the user side of the framework, and the details of our concrete prototype implementation.

3.1 The Aggregator Side

The framework places relatively little constraint on the data aggregator other than to prescribe the format of queries. The design of the mechanism and its practical utility are up to the aggregator. In order for the aggregator to have any chance of assessing the utility of the mechanism (e.g. the expected error) we assume that the aggregator has some knowledge (exact or approximate) of the privacy policy that the users will enforce (which will typically mean a privacy budget). It is therefore in the interest of the aggregator not to ask queries which are over-budget (as this will result in, at best, rejection). The aggregator also has the responsibility to give a realistic upper bound on the processing time for the query. This enables the pre-processing of the query to be implemented as a constant-time operation, thus avoiding the transmission of sensitive information on a covert timing channel. Again, it is in the interest of the aggregator to give a good upper bound, otherwise the response from users will be pure noise.

Many mechanisms rely on more than one round of communication between the aggregator and the users, but this is unproblematic. The use of personalised budgets makes the process of decomposing queries or querying a selected subset of users straightforward, although this may require (for the sake of good utility) that the aggregator performs their own bookkeeping to track the personalised privacy budgets of users.

3.2 The User Side

The aggregator targets users and tunes the parameters in such a way that it complies with the aggregator's expectation of the user privacy policy. However, this is not enough to guarantee that the aggregator does not breach the individual's privacy by mistake or the aggregator does not act maliciously. The framework on the user side is responsible for the correct enforcement of the intended privacy policy. The core of the method is that it is sufficient for the user to check the cost of the randomised response matrix T to determine the privacy of the query. We temper this with the addition of two local policy functions, which give a flexible way for the user to choose whether to respond to a query or reject it.

Recall that the user has a local state local, which comprises two parts, local.private inaccessible to the aggregator, and local.public which we do not attempt to hide from the aggregator, for the private and public data, respectively. We assume at least the following subfields of the state:

- local.public.id, the identity (index) of the user, assumed to be invariant,
- local.public.policy, which is the public policy, a predicate on the query which determines whether the query will be accepted or rejected. This function also has read access to all the local public information local.public, but not the private data. Since the function is assumed to be pure then the result of the predicate is not sensitive.
- local.public.totalCost records the actual privacy cost (the local epsilon) which has been incurred so far,
- local.private.policy, a private policy, potentially implementing a stricter policy than the public one, but in a way which is potentially sensitive; this pure function has access to the whole of the local state.
- local.private.data, the sensitive data to which the query computation will be applied,
- local.public.data, the environment that stores local non-sensitive data, including the query history
- local.public.reject, determines the response to be sent when a query is publicly rejected.
- local.public.update, determines how the local public state will be updated after responding to the query.

These components are used by the user as depicted in Algorithm 1, and explained in more detail below.

We assume that the functions in the local state are all pure (they can only read the parameters provided, and cannot modify the local state or communicate in any other way).

The sandbox function is a timing sandbox (in the actual implementation discussed in Section 3.4 it is also a sandbox preventing arbitrary side-effects) which ensures that $sandbox(t, f, v)$ delivers a result after t

Algorithm 1: User Procedure

Input : $Q = \langle t, pre, T, post \rangle$ from aggregator A such that *pre* is the pre-processing function, t the pre-processing time-out, T the stochastic matrix, and *post* is the post-processing function

```
1  if local.public.policy(Q, local.public) then
2      local.public.totalCost = cost(T) + local.public.totalCost
3      preIn = sandbox(c,
4      function constantExec()
5          r = a random data item from domain of local.private.data
6          if local.private.policy(Q, local)
7           then
8              return data
9          else
10             return r
11         end
12     , ( ) )
13     preOut = T(sandbox(t, Q.pre, preIn))
14     result = sandbox(∞, Q. post, preOut)
15 else
16     result = local.public.reject(Q, local.public.data)
17 end
18 local.public.data= local.public.update (local.public.data, Q, result)
```

Reply: $\langle$local.public.id, result)$\rangle$ to A

time units, returning the result of $f(v)$ if it completes in time, and giving a dummy valueof the same domain otherwise, similar to the approach used by the Fuzz system [20] to enforce a fixed amount of time for a microquery.

Differential Privacy of the User Procedure The core of the algorithm is the simple composition of functions $post \circ T \circ pre$. Assuming that the initial value of local.public.totalCost is zero, we argue that the value of this variable provides the amount of local differential privacy that the algorithm has at any point in time. But this computation is mediated by two policy checks, a public policy and a private policy.

The public policy decides whether to accept or reject the query based purely on the local public data and the query. Since this predicate does not access sensitive data its output and running time are not sensitive. In the case that the query is rejected (for example, if the query would cause the local privacy budget to be exceeded) then a reject function is called. This can be used to send a rejection notice to the aggregator, or even send a cached value of an earlier query response (similar to what is done in Rappor [16]).

Once a query has been "officially accepted" then the privacy cost is recorded in the public state (line 2). In the next step an optional private

policy is applied. The private policy can depend on any part of the state, so the outcome is secret. The purpose here is to give the possibility to impose a policy which is more fine-grained than differential privacy. In the case that the private policy decides to reject the query then the pre-processor is fed with a dummy value instead of the actual data. The correctness of this step follows from the pre-processing theorem since the choice of the real value vs a dummy value can be viewed as an additional pre-processing function. Whatever the result is, it is transmitted to the aggregator and recorded in the public data with the restricted update function (Line 18).

To keep the algorithm simple, it is assumed that the user-defined policies are not malicious, and respect the implicit restrictions that are placed (i.e. that they are pure functions). If this were not the case then the public policy might access the private data and leak information directly or indirectly. Furthermore, the execution time of the condition block (line 3-12) on the private data has to be constant. To enforce all mentioned restrictions, policies and the private policy condition block are executed in a sandbox with a constant execution time of c.

The differential privacy guarantee of local.public.totalCost for the result of $T \circ sandbox$ is immediate from Proposition 2. Line 13 in the Algorithm 1 is simplified with the assumption that the *pre* produces values in the domain of the matrix. Effectively the pre-processing, *pre* , is supplemented with a clamping wrapper that maps a result of the pre-processing into the stochastic matrix domain. Utilising Theorem 2 for the private privacy policy, the pre-processing and the clamping wrapper, it is evident that they altogether have no privacy impact. Finally, the post-processing (line 14) with no access to the private data has already been shown [12] to be privacy harmless.

3.3 Modelling Different Policies

Modular design and flexible building blocks such as a generic personalised user policy allow the extension of the framework with a variety of differential privacy mechanisms and modelling various flavours of differential privacy. The framework allows users to choose public privacy policies that are (in principle) accessible to the aggregator. In some circumstances one might argue that the privacy policy is a sensitive object and it is not necessarily data independent. This argument is supported with the "Nothing to hide argument", that those who have something to hide might choose stricter privacy policies. This choice of policy may bias the overall result toward responses from individuals with less sensitive data and respectively less strict privacy policies. The presence of a private policy gives the option to refine differential privacy with a fine-grained decision potentially based on sensitive data.

Simple Personalised Differential Privacy As an example of the most basic public policy function, we assume that there is a part of the local state containing the non-sensitive privacy budget, local.public.budget. Then we could define the simplest "reasonable" policy to enforce budget as:

$$\begin{aligned}&\mathsf{local.public.policy}(\langle pre, t, T, post\rangle, \mathsf{local.public})\\ &\qquad = cost(T) + \mathsf{local.public.totalCost} \leq \mathsf{local.public.budget}\end{aligned}$$

The simplest way to ensure that the budget is not sensitive, and to guarantee a meaningful global differential privacy property, is to assume that all users start out with the same initial budget (but perhaps at different times).

The existence of a private policy allows a more fine-grained but potentially private-data dependent. From the aggregator perspective the utility of the analysis in the presence of a private policy may be questionable; we do not argue that they are an important feature, only that they can be modelled.

Event Level Differential Privacy[14] Policy functions make it possible to represent *event level differential privacy* on a data stream. Event level differential privacy allows only limited information to be extracted from a data stream during a particular period, to hide the value of an event. The method is an attractive choice when correlation between user's data in different time points is limited or negligible. This is most simply viewed as running a "new" framework instance for each event. Alternatively it can be modelled by a public policy which maintains a budget for each event.

Heterogeneous Privacy[4] Heterogeneous differential privacy allows users to label different fields of data with different sensitivities. One way to view heterogeneous differential privacy in our framework is to assume that the query recognisably targets a particular part of the data. We then maintain a public or private count of the privacy cost for each field of the data. When the cost would exceed the local budget for any field the query will be rejected. The choice of private vs public policy would reflect the way in which the initial budget is set (e.g. if it is set by a third party then it may be acceptable to view the budget as public, but if it is set by the user the values may reveal information about the values of the fields). Note that the overall local differential privacy will be guaranteed to be no greater than the sum of the respective budgets of all the fields, so this can be combined with a public overall budget.

Permanent and Instantaneous Randomised Response[16] RAPPOR introduces a two-step randomisation technique. It assumes a fixed query, and enforces a kind of event-level privacy for user values, where each time

there is a new value, a "permanent" randomised response is memorized for that value; for the lifetime of that value, every time there is a query a further randomisation of the value is applied. The purpose of this second round of randomisation is to make it difficult for the aggregator to correlate responses over time. For this to be useful, however, the aggregator must not track the id of the sender. To implement this without having to trust the sender would need a trusted mix network between the clients and the server. We can certainly implement a policy which caches the results of a query to replay (at no extra cost) later, but it is less obvious how to re-randomise an already differentially private value in our framework. This is not surprising since our framework was only designed for local differential privacy. The purpose of the second randomisation in RAPPOR goes beyond this goal in some way which is not precisely specified.

3.4 Implementation

In this section we discuss our prototype implementation [3], with focus on the assumptions made and how the implementation satisfies the assumptions. The prototype of the framework is implemented in Python and uses operating system features for sandboxing. In the basic setting each user runs a lightweight HTTP server and the aggregator communicates with the user by uploading an analysis and getting the result back as an XML document.

Process Isolation The pre-processing and post-processing programs sent to a user are executed in a sandbox to avoid direct access to sensitive data. Additionally, the user-specified policies are executed in a sandbox to ensure its data-independency.

We use the Nsjail[19] sandboxing tool that takes advantages of (Linux) operating system security features such as namespaces, *seccomp-bpf* and *cgroups* to isolate a process. For executing the pre-processing, we have assumed that the behaviour of the pre-processor is functional - i.e. side-effect free. To ensure the validity of this assumption, the sensitive data and the analysis data are mounted in a read-only file system and a writeable file system is mounted for the output. While virtualisation solutions generally provide better process isolation, for our prototype implementation we chose the more light-weight Nsjail sandbox. Although timing is handled by this tool, we are aware of other possible side channels (e.g. through memory cache) in both sandboxing and virtualisation solutions that need to be considered for the finished product. To achieve parallel execution of two sandboxes, with no communication between sandboxes,

[3] The implementation is available at `https://drive.google.com/file/d/0By2ilpP5pdV_Wnhwems5Z1V3NFE/view?usp=sharing`

mutual exclusion of resource usage (e.g. memory and CPU time) would need to be guaranteed in advance.

Timing Channel Nsjail supports the control over execution time that seems suitable for our implementation: it terminates any program that takes longer than a given time bound, and it ensures that the result will not be available before the specified time by padding the execution time up to the bound. Having said that, we have not studied whether the time guarantees are sufficiently precise in the presence of an adversary looking at small timing variations.

Domain Enforcement Algorithm 1 assumes that output of pre-processing is in the expected domain. In reality it may produce an output outside the expected domain or produce an error (the sandbox takes care of the non-terminating case). The clamping procedure is placed in between these two procedures ($T \circ \text{clamp} \circ \textit{sandbox}$) to capture exceptional cases. The clamping procedure works as an identity function ($T \circ \text{id} \circ \textit{sandbox}$) except in the following cases:

- producing a result out of the domain,
- errors (e.g. lack of memory or arithmetic errors), or
- reaching the time-out and not producing a result.

These special cases have to be either 1) captured as one of the cases in the stochastic matrix 2) mapped to a random item in the domain (the domain can be extracted from T).

Optimisations A stochastic matrix for randomizing a bit string of size d bits would have a total size of $2^{(2d)}$. It is clearly infeasible to transmit as part of a query for anything but tiny values of d, so optimising the representation of such a structure is essential. Fortunately actual algorithms in almost all cases we have come across (i) randomise each bit independently, and (ii) use the same randomisation for each bit (see Section 4). Point (i) means that we need only d two-by-two matrices (i.e. $\mathcal{O}(d)$ data) to represent the transformation, and the cost is calculated as the maximum of the costs of the d matrices. But feature (ii) reduces the space and cost calculation further to $\mathcal{O}(1)$, since only a single two-by-two matrix need be transmitted. Additionally, the parties can further reduce the matrices communication cost by agreeing on some standard families of matrices with a pre-defined format represented by the matrix type and its privacy parameter (e.g. *staircase matrix [28] with* $\varepsilon = 0.2$).

3.5 Deployment

As a scenario for the use of PreTPost, we considered an example of an Internet Service Provider (ISP) aiming to study the overall security level of

their users. The analyst works for the ISP and the user data is collected from routers in private homes. The analysis uses sensitive parameters such as the time that the password is changed, the usage of network segmentation, the encryption type and the usage of MAC address filtering.

We ported the user side of the framework to embedded devices by cross-compiling the needed utilities to make them executable on the OpenWRT Linux distro. The OpenWRT project and its successor the LEDE project are aiming to provide a light-weight Linux distribution for embedded devices and routers with limited storage capacity and computation power. The communications between users and the curator are protected by SSL encryption and the permission to perform queries is restricted to the curator by access credentials in the form of username and password. Every individual user decides about the level of privacy that will be used for the budget enforcement policy, as well as which resources will be accessible to the curator. Since the budget values are not necessarily uniform, the curator may use different strategies to decide on the privacy cost for individual analyses sent to users. In this case the curator may use a cost proportional to the remaining budget rather than an identical cost for all analyses. Optionally the curator may use an identical query for a subset of the users that agree on the analysis and its privacy cost.

4 Case Studies

In this section we outline the examples from the literature which we have successfully rewritten to the format of the framework, and give a high-level outline of the process of expressing an algorithm in the form explained in Section 3.

In the *pre-processing* phase, sensitive user data in different forms such as time series data points, graphs or tabular data is gathered from internal sensors or directly from users (in the form of a survey). Different operations such as collection, transformation (computation on the values), encoding values into bit vector and hashing [16], dimensionality reduction and addition of error correction code [7], aggregation and categorization of data or any other deterministic operation can all be done in this phase.

The best privacy practice is to take out unnecessary information and to aggregate, compress and reduce the precision as much as possible to ensure data minimization principles [12]. As visible from the formulation of differential privacy, keeping the size of a report short can reduce the privacy cost and improve accuracy.

The privacy preserving *probabilistic* part of an algorithm that is applied to the intermediate result is modelled in the transformation matrix. Finally, what remains from the analysis (such as packaging the

anonymised results or interactions between the user and the aggregator) is modelled in the post-processing phase. Note that post-processing solely has access to the result of the randomisation matrix and public data, therefore this optional phase has no privacy impact. Post-processing can reduce the communication cost when queries are adaptive and helps to remain faithful to the shape and format of reports for existing algorithms.

Algorithm 2: $\mathcal{R}$: ε-Basic Randomiser[7]

Input : ε, m-bit string $x \in \{-1/\sqrt{m}, 1/\sqrt{m}\}^m \cup \{0\}$

1 Sample uniformly $j \leftarrow [1 \dots m]$;

2 **if** $x \neq 0$ **then**

3 Randomise j-th bit x_j of the input x

$$z_j = \begin{cases} m \cdot c_\varepsilon x_j \text{ with the probability} \frac{e^\varepsilon}{1+e^\varepsilon} \\ -m \cdot c_\varepsilon x_j \text{ with the probability} \frac{1}{1+e^\varepsilon} \end{cases}$$

where $c_\varepsilon = \frac{e^\varepsilon+1}{e^\varepsilon-1}$

4 **else**

5 Sample uniformly $z_j \leftarrow \{-c_\varepsilon\sqrt{m}, c_\varepsilon\sqrt{m}\}$;

6 **end**

Output: (j, z_j)

Let us begin with considering a building block for more complicated algorithms. The $\mathcal{R}$-Basic Randomiser[7] presented in the Algorithm 2, takes a bit string of length n to produce a differentially private report. The Algorithm 2 is placed in a separate function to make it simpler to only consider the privacy aspect of the more complicated algorithm.

Note that the output of the algorithm includes j, a random sample generated in line 1. From the point of view of differential privacy it might be reasonable to view this with some suspicion. A differentially private algorithm is certainly randomised, but it must not reveal the randomness that it uses in an essential way.

As we will show in the rest of the section, while proving every individual algorithm is tedious, de-constructing it into a pre-processing algorithm, a stochastic matrix , and a post-processing algorithm as demonstrated in Algorithm 3 and using pre-post processing arguments makes correctness arguments trivial – which is of course why the framework is able to automate it.

The decomposition of Algorithm 2 into the form required by the framework is shown in Algorithm 3. Recall that in our definition of a local mechanism L we in fact we have a family of functions $\{L(P, \_)\}$ one for each public input P. The key to representing Algorithm 2 in the

Algorithm 3: Decomposition of Basic Randomiser

Function `Pre`(ε, $x \in \{-1/\sqrt{m}, 1/\sqrt{m}\}^m \cup \{0\}$)**:**

Sample $j \leftarrow [1 \dots m]$ **with the seed** r
$c_\varepsilon = \frac{e^\varepsilon + 1}{e^\varepsilon - 1}$
if $x \neq 0$ **then**
  $z_j \leftarrow m \cdot c_\varepsilon \cdot x_j$
else
  Sample $z_j \leftarrow \{-c_\varepsilon\sqrt{m}, c_\varepsilon\sqrt{m}\}$
end
return z_j

$$T = \begin{array}{c} \\ \frac{-1}{\sqrt{m}} \\ \frac{1}{\sqrt{m}} \end{array} \begin{array}{c} \begin{array}{cc} \frac{-1}{\sqrt{m}} & \frac{1}{\sqrt{m}} \end{array} \\ \begin{pmatrix} \frac{e^\varepsilon}{1+e^\varepsilon} & \frac{1}{1+e^\varepsilon} \\ \frac{1}{1+e^\varepsilon} & \frac{e^\varepsilon}{1+e^\varepsilon} \end{pmatrix} \end{array}$$

Function `Post`(z)**:**

Sample $j \leftarrow [1 \dots m]$ **with the seed** r
return (j, z)

framework is to make explicit the fact[4] that the random value j is not secret, but part of the public input. In the representation of this we refer to this entropy source as a seed r from which a value is deterministically generated.

Other than the treatment of the random value, the pre-processing and post-processing algorithms share most of their code with the reference algorithm (Algorithm 2).

The high level algorithms [7, 8, 9] that use the ε-basic randomiser ($\mathcal{R}$) essentially just add a specific pre-processing phase and scale the method across n bits.

To demonstrate the effectiveness of the proposed structure for writing queries introduced in Section 3 we collected available local algorithms in the discrete domain and decomposed and integrated them into the framework. The methods we have encoded are listed below:

Warner's Method (Randomised Response) The basic idea of unbiased coin flip can be used to anonymise a predicate. The privacy parameter is adjustable by making the coin biased. For categories

[4] Bassily and Smith (loc cit) note: "the randomness in the choice of j may come from outside the randomizer: it could be sent by the server, available as public coins, or generated pseudorandomly from other information"

with more than two members one simply changes the dimensions of the transformation matrix.

Erlingsson et al.'s One-time method (RAPPOR) RAPPOR [16] is developed by Google and integrated into the Chrome browser to get statistics from Internet usage behaviour and browser configuration by encoding strings (e.g. browsers home-pages) in a bloom filter and randomizing it. The size of the bloom filter, the number of hashes and the number of cohorts (users that use similar hashes for their bloom filter) are relevant parameters for this method. RAPPOR's method is especially useful when we are dealing with statistics on strings that are not enumerable in advance like URLs, domain names and process names. While the original idea behind RAPPOR was mostly to identify distribution of a single variable(string) ; it was later extended [17] to study associations and correlations between multiple variables, especially when the dictionary of possible string values is not known. Modelling the core of RAPPOR – the part which has the goal of achieving local differential privacy called *one-time RAPPOR* – is straightforward in our framework. As discussed in Section 3.3, the most general RAPPOR method contains a memoisation of earlier results, coupled with a second layer of randomisation which is targeted at preventing correlation of the responses over time which facilitate "tracking".

We believe we can model this using a slight variant of their algorithm, but since the privacy goals of this second randomisation and the composition theorem are not precisely specified, it is difficult to establish whether our variant is as intended.

Bassily and Smith's Method The method explained by Bassily and Smith[7] uses a Johnson-Lindenstrauss transformation to encode a large domain in a smaller dimension and uses a basic randomiser to construct a frequency oracle.

Duchi et al.'s Method Central tendency or measuring mean value is another statistical parameter that gives a good overall view of data. Duchi et al. [10] proposed an algorithm for handling of mean in numerical values that was later claimed by Nguyên et al. [33] to violate differential privacy. In the framework we implemented the fixed algorithm proposed by Nguyên et al..

Nguyên et al.'s Method (Harmony) Nguyên et al. propose another algorithm that outperforms Duchi et al.'s Method. Note that Akter and Hashem[3] use a similar method based on Bernoulli probability distribution to handle numeric data which are all similarly integrable in the framework.

The above case studies, similar to the decomposition of Algorithm 2, generally have a simple pre-processing phase with a more complicated post-processing phase. Mainly a single 2×2 matrix can be used repeatedly for representing the randomised responses matrix T, and thus the

optimisation discussed in Section 3.4 avoids transmission of more than a constant amount of data.

5 Utility Issues

This section discusses the effect of user-defined policies and personalised differential privacy on utility. We argue how curator's capability to choose the sample space and privacy parameters and the users freedom to reject an analysis that doesn't comply with their expectation effect the utility.

In differential privacy, choosing the ε value is an intrinsically challenging task[24]. High values for ε provide little privacy and low values give the analyst no meaningful results which make the entire process of data collection pointless. As a part of the data minimization principle, the aggregator, with a look into the future, does not collect more data than needed which means that the choice of ε should be done based on the expected accuracy (resp. error).

The benefits of using local differential privacy as explained in Section 1 do not come without cost. First and foremost, not all analyses are achievable with the same accuracy compared to the central model. Additional computation and communication resources may be needed for collecting, transmitting and processing the data. For several analyses additional parameters (such as the number of participants willing to contribute) and the type of computation have to be known before data collection.

Theoretical utility functions for mutual information and hypothesis testing are studied by Kairouz et al.[28]. The common utility-accuracy metrics such as f-divergence (for hypothesis testing), mutual information (for information preservation) and expected standard error are studied in several studies. In this section we look into parameters that affect the utility and how the aggregator, in charge of accuracy, can benefit from choosing a right strategy.

Being aware of the users' policies helps the aggregator to sample the minimum number of participants and to adjust the parameters so as to achieve the minimum required accuracy. By negotiating privacy parameters with each user (alternatively querying the public privacy policies of the users) the aggregator estimates the number of participants, the proper privacy parameters and the overall error. In case the result would be more accurate than required the aggregator has the chance to adjust the parameters before query execution. If the level of accuracy is not achievable the query does not need to be launched at all.

This pre-analysis procedure assures that the eventual reports the aggregator gets from users are not more accurate than needed and at the same time not too noisy for the result to be untrustworthy.

The aggregator A samples and chooses the working dataset based on the public background information (e.g. geolocation area) and the results

from previous queries. The aggregator has to consider that not all previous responses are truthful when they are used for the upcoming queries. Sampling by giving each user the choice of participation is discussed in [27].

6 Related Work

On Centralised Differential Privacy Frameworks A number of frameworks support the construction of correct differentially private mechanisms. The PINQ framework, [30, 31], was one of the first, and provides a C# API for accessing a sensitive database which tracks privacy costs using general composition principles (e.g. performing an ε differentially-private query after a low sensitivity such as database selection is still ε differentially-private), and ensures that a fixed privacy budget is never exceeded. Later variants include wPINQ [36] using the idea of adjusting the sensitivity of data operations by weighting the contributions from individual records accordingly, streaming-PINQ[46] for streaming data, and the *ProPer* system [15] which uses provenance tracking and personalised budgets for better budget utilisation. PINQ-like systems are all frameworks with run-time enforcement of differential privacy. The GUPT[32] framework leverages sample-and-aggregate [34] and introduces a novel notion of degradation of data sensitivity over time. Airavat[39] combines a relatively simple differentially private MapReduce framework with mandatory access control to remove the need for auditing potentially malicious code.

Fuzz and DFuzz [20, 37, 18] use similar underlying principles to PINQ, but use a custom language with a sophisticated type system centred around the concept of function sensitivity, to enable the verification of differential privacy by type-checking. A recent variant extends this to approximate differential privacy with a hybrid static-dynamic approach [47].

For relational algebra that is the heart of modelling relational databases, Palamidessi and Stronati[35] introduce a method to measure the sensitivity of a query in a compositional way and FLEX[25] is a framework to enforce differential privacy on the SQL query level.

There are also frameworks which support the semi-automated verification of differential privacy from first principles. Barthe et al.[6] introduce a probabilistic relational Hoare-logic for formal manual verification of differential privacy of algorithms using a tool based on Coq. Zhang and Kifer [49] introduce a more lightweight verification method which is able to verify some sophisticated algorithms with a relatively small amount of user input.

On Local Differential Privacy Frameworks Using distortion matrix to perturb values for a privacy preserving data mining is a pre-differential

privacy technique[38, 2]. In the local differential privacy model there is no prior work, to our knowledge, in the spirit of the framework of the present paper. In the local model we have seen integration of differential privacy in Apple's iOS 10 suggests emojis and words, which according to a recent patents [42, 43] includes a client budget. As discussed, RAPPOR[16] for the Chrome web browser implements differentially private methods to report on common settings and home pages. Harmony[33] aims to do something similar for mobile devices. RRreg[22] is an "R" package to analyse correlation and regression for randomised responses.

Personalised Differential Privacy The concept of personalized anonymity is first introduced by Xiao and Tao [48] to let individuals specify the *k-anonymity* of their sensitive values. Personalised differential privacy as a generalisation of standard differential privacy was introduced around the same time in two independent works [15] and [26], both for the centralised model. Chen et al. [8] introduce a notion of *personalized local differential privacy* which also includes a weakening of the requirements of differential privacy whereby users specify not only a personal budget, but a specification of which subset of the value range should be protected (e.g. I choose only to require epsilon differential privacy for my salary if it is over 100K/year). Alaggan et al. [4] propose the concept of *heterogeneous differential privacy* – a variant of local differential privacy in which different fields of the user's data record can be assigned different budgets.

7 Future Work

Utility Tools Our emphasis in this work has been on privacy from the perspective of the data owner. We have focussed on user's ability to verify the privacy properties of dynamic queries. Future work includes complementary aggregator-centric tools to help an aggregator design queries with good utility. The aggregator cannot simply try different algorithms on the sensitive dataset since the privacy budget is limited, and the aggregator has no access to the ground truth. This limitation emphasises the importance of evaluation of each algorithm before empirical use. In the centralized model, DPcomp[21] evaluates the performance of a differential privacy algorithm in sample datasets while considering several parameters, especially privacy loss ε, the dataset shape, domain size and scale. In the local setting, in addition to these parameters, the algorithm may use extra parameters to tune the algorithm. Other arguments and assumptions such as knowing the number of participants before performing the analysis makes the comparison of the methods even more challenging.

Laplace-based Mechanisms Our framework exclusively focussed on discrete randomisation methods. In the centralised model differential privacy and the Laplace noise mechanism are historically introduced hand

in hand. Algorithms that take advantage of Laplace noise instead of working in a discrete domain [7, 23] or other basic mechanisms can easily be integrated in the framework, and our prototype implementation supports this, although the meta-theory in the form of a more general proof of the pre-processing theorem (Theorem 2) has not yet been worked out. The comparison of utility preservation between the Laplace mechanism and a randomised response is done by Wang et al. in [44].

Language-based Implementation We have relied on sandboxing and a very simple fixed query structure to realise our framework. An alternative would be to use programming-language-level guarantees, for example leveraging safe Haskell [41] for giving purity guarantees, or even custom type systems in the style of [18] to allow more liberal program structures than just $post \circ T \circ pre$.

8 Conclusion

Centralised differential privacy requires a high level of trust in the data aggregator to keep the data secure and to apply the correct algorithms. The recent adaptation of local differential privacy[5, 16] is a step toward better privacy for users. However, implementations are not transparent and as shown by Tang et al.[40] they can be promiscuous in the choice of the privacy parameter epsilon.

Compared to the central setting in which the curator has the flexibility to perform the desired analysis, in the current systems the analyses are hard-coded. Sending a new set of analyses in the next system update is possible but verification of these arbitrary analyses is nontrivial. The framework allows the user to verify the differential privacy of an incoming query without trusting the analyst. This is achieved by a framework with isolated components (Pre, T, Post) that communicate with each other in a restricted setting. This decomposition allows the client to verify the privacy harm of analyses and avoid a malicious analysis that exfiltrates sensitive data from the user without anonymisation.

We have studied existing local differential privacy algorithms in the literature and found that all are decomposable to be integrated in the PreTPost framework. We further demonstrate that rewriting the algorithms within the PreTPost framework ensures that not only the theoretical algorithms but also their implementations are not prone to side channels.

References

[1] Nabil R. Adam and John C. Worthmann. Security-control methods for statistical databases: A comparative study. *ACM Comput. Surv.*, 21(4):515–556, December 1989. ISSN 0360-0300.

[2] Shipra Agrawal, Jayant R. Haritsa, and B. Aditya Prakash. Frapp: a framework for high-accuracy privacy-preserving mining. *Data Mining and Knowledge Discovery*, 18(1):101–139, Feb 2009. ISSN 1573-756X.

[3] Mousumi Akter and Tanzima Hashem. *Computing Aggregates Over Numeric Data with Personalized Local Differential Privacy*, pages 249–260. Springer International Publishing, Cham, 2017.

[4] Mohammad Alaggan, Sébastien Gambs, and Anne-Marie Kermarrec. Heterogeneous differential privacy. *Journal of Privacy and Confidentiality*, 7, Issue 3, 2016.

[5] Apple Press Release. Apple previews ios 10, the biggest ios release ever, 2016.

[6] Gilles Barthe, Boris Köpf, Federico Olmedo, and Santiago Zanella Béguelin. Probabilistic relational reasoning for differential privacy. In *Proceedings of the 39th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '12, pages 97–110, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1083-3.

[7] Raef Bassily and Adam Smith. Local, private, efficient protocols for succinct histograms. In *Proceedings of the Forty-seventh Annual ACM Symposium on Theory of Computing*, STOC '15, pages 127–135, New York, NY, USA, 2015. ACM. ISBN 978-1-4503-3536-2.

[8] R. Chen, H. Li, A. K. Qin, S. P. Kasiviswanathan, and H. Jin. Private spatial data aggregation in the local setting. In *2016 IEEE 32nd International Conference on Data Engineering (ICDE)*, pages 289–300, May 2016.

[9] J. C. Duchi, M. I. Jordan, and M. J. Wainwright. Local privacy and statistical minimax rates. In *2013 51st Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pages 1592–1592, Oct 2013.

[10] John C. Duchi, Michael I. Jordan, and Martin J. Wainwright. Local privacy and minimax bounds: Sharp rates for probability estimation. In *Proceedings of the 26th International Conference on Neural Information Processing Systems*, NIPS'13, pages 1529–1537, USA, 2013. Curran Associates Inc.

[11] Cynthia Dwork. Differential privacy. In *Automata, languages and programming*, pages 1–12. Springer, 2006.

[12] Cynthia Dwork and Aaron Roth. The algorithmic foundations of differential privacy. *Found. Trends Theor. Comput. Sci.*, 9:211–407, August 2014. ISSN 1551-305X.

[13] 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.

[14] Cynthia Dwork, Moni Naor, Toniann Pitassi, and Guy N. Rothblum. Differential privacy under continual observation. In *Proceedings of the Forty-second ACM Symposium on Theory of Computing*, STOC '10, New York, NY, USA, 2010. ISBN 978-1-4503-0050-6.

[15] Hamid Ebadi, David Sands, and Gerardo Schneider. Differential privacy: Now it's getting personal. In *Proceedings of the 42Nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '15, pages 69–81. ACM, 2015. ISBN 978-1-4503-3300-9.

[16] Úlfar Erlingsson, Vasyl Pihur, and Aleksandra Korolova. RAPPOR: Randomized aggregatable privacy-preserving ordinal response. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, CCS '14, New York, NY, USA, 2014. ISBN 978-1-4503-2957-6.

[17] Giulia C. Fanti, Vasyl Pihur, and Úlfar Erlingsson. Building a RAPPOR with the unknown: Privacy-preserving learning of associations and data dictionaries. *PoPETs*, 2016:41–61, 2016.

[18] Marco Gaboardi, Andreas Haeberlen, Justin Hsu, Arjun Narayan, and Benjamin C. Pierce. Linear dependent types for differential privacy. In *Proceedings of the 40th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '13, pages 357–370, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-1832-7.

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

[20] Andreas Haeberlen, Benjamin C. Pierce, and Arjun Narayan. Differential privacy under fire. In *Proceedings of the 20th USENIX Conference on Security*, SEC'11, pages 33–33, Berkeley, CA, USA, 2011. USENIX Association.

[21] Michael Hay, Ashwin Machanavajjhala, Gerome Miklau, Yan Chen, Dan Zhang, and George Bissias. Exploring privacy-accuracy tradeoffs using dpcomp. In *Proceedings of the 2016 International Conference on Management of Data*, SIGMOD '16, pages 2101–2104, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-3531-7.

[22] Daniel W. Heck and Morten Moshagen. *RRreg: Correlation and Regression Analyses for Randomized Response Data*, 2017. R package version 0.6.2.

[23] Justin Hsu, Sanjeev Khanna, and Aaron Roth. *Distributed Private Heavy Hitters*, pages 461–472. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012. ISBN 978-3-642-31594-7.

[24] Justin Hsu, Marco Gaboardi, Andreas Haeberlen, Sanjeev Khanna, Arjun Narayan, Benjamin C. Pierce, and Aaron Roth. Differential privacy: An economic method for choosing epsilon. In *Proceedings of the 2014 IEEE 27th Computer Security Foundations Symposium*,

CSF '14, pages 398–410, Washington, DC, USA, 2014. IEEE Computer Society. ISBN 978-1-4799-4290-9.

[25] Noah Johnson, Joseph P. Near, and Dawn Song. Practical differential privacy for sql queries using elastic sensitivity. *CoRR*, abs/1706.09479, 2017.

[26] Z. Jorgensen, T. Yu, and G. Cormode. Conservative or liberal? personalized differential privacy. In *2015 IEEE 31st International Conference on Data Engineering*, pages 1023–1034, April 2015.

[27] Joshua Joy and Mario Gerla. Anonymized local privacy. *CoRR*, abs/1703.07949, 2017.

[28] Peter Kairouz, Sewoong Oh, and Pramod Viswanath. Extremal mechanisms for local differential privacy. *Journal of Machine Learning Research*, 17(17):1–51, 2016.

[29] Daniel Kifer and Ashwin Machanavajjhala. No free lunch in data privacy. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data*, SIGMOD '11, pages 193–204, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0661-4.

[30] Frank McSherry. Privacy integrated queries. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data (SIGMOD)*. Association for Computing Machinery, Inc., June 2009.

[31] Frank McSherry. Privacy integrated queries: an extensible platform for privacy-preserving data analysis. *Commun. ACM*, 53(9):89–97, 2010.

[32] Prashanth Mohan, Abhradeep Thakurta, Elaine Shi, Dawn Song, and David Culler. Gupt: Privacy preserving data analysis made easy. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, SIGMOD '12, New York, NY, USA, 2012. ISBN 978-1-4503-1247-9.

[33] 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.

[34] Kobbi Nissim, Sofya Raskhodnikova, and Adam D. Smith. Smooth sensitivity and sampling in private data analysis. In David S. Johnson and Uriel Feige, editors, *STOC*, pages 75–84. ACM, 2007. ISBN 978-1-59593-631-8.

[35] Catuscia Palamidessi and Marco Stronati. Differential privacy for relational algebra: Improving the sensitivity bounds via constraint systems. In *Proceedings 10th Workshop on Quantitative Aspects of Programming Languages and Systems*, 2012.

[36] Davide Proserpio, Sharon Goldberg, and Frank McSherry. Calibrating data to sensitivity in private data analysis: A platform for differentially-private analysis of weighted datasets. *Proc. VLDB Endow.*, 7(8):637–648, April 2014. ISSN 2150-8097.

[37] Jason Reed and Benjamin C. Pierce. Distance makes the types grow stronger: A calculus for differential privacy. In *Proceedings of the 15th ACM SIGPLAN International Conference on Functional Programming*, ICFP '10, pages 157–168, New York, NY, USA, 2010. ACM. ISBN 978-1-60558-794-3.

[38] Shariq J. Rizvi and Jayant R. Haritsa. Maintaining data privacy in association rule mining. In *Proceedings of the 28th International Conference on Very Large Data Bases*, VLDB '02, pages 682–693. VLDB Endowment, 2002.

[39] Indrajit Roy, Srinath Setty, Ann Kilzer, Vitaly Shmatikov, and Emmett Witchel. Airavat: Security and privacy for mapreduce. In *Symposium on Networked Systems Design and Implementation (NSDI)*. USENIX - Advanced Computing Systems Association, April 2010.

[40] Jun Tang, Aleksandra Korolova, Xiaolong Bai, Xueqiang Wang, and XiaoFeng Wang. Privacy loss in apple's implementation of differential privacy on macos 10.12. *CoRR*, abs/1709.02753, 2017.

[41] David Terei, Simon Marlow, Simon Peyton Jones, and David Mazières. Safe haskell. In *Proceedings of the 2012 Haskell Symposium*, Haskell '12, pages 137–148, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1574-6.

[42] A.G. Thakurta, A.H. Vyrros, U.S. Vaishampayan, G. Kapoor, J. Freudiger, V.R. Sridhar, and D. Davidson. Learning new words, 2017. US Patent 9,594,741.

[43] A.G. Thakurta, A.H. Vyrros, U.S. Vaishampayan, G. Kapoor, J. Freudinger, V.V. Prakash, A. Legendre, and S. Duplinsky. Emoji frequency detection and deep link frequency, July 11 2017. US Patent 9,705,908.

[44] Yue Wang, Xintao Wu, and Donghui Hu. Using randomized response for differential privacy preserving data collection. In *EDBT/ICDT Workshops*, 2016.

[45] Stanley L. Warner. Randomized response: A survey technique for eliminating evasive answer bias. *Journal of the American Statistical Association*, 60(309):63–69, 1965. ISSN 01621459.

[46] Lucas Waye. Privacy integrated data stream queries. In *Proceedings of the 2014 International Workshop on Privacy & Security in Programming*, PSP '14, New York, NY, USA, 2014. ISBN 978-1-4503-2296-6.

[47] Daniel Winograd-Cort, Andreas Haeberlen, Aaron Roth, and Benjamin C. Pierce. A framework for adaptive differential privacy. In *icfp17*, 2017.

[48] Xiaokui Xiao and Yufei Tao. Personalized privacy preservation. In *Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data*, SIGMOD '06, pages 229–240, New York, NY, USA, 2006. ACM. ISBN 1-59593-434-0.

[49] Danfeng Zhang and Daniel Kifer. LightDP: towards automating differential privacy proofs. In Giuseppe Castagna and Andrew D. Gordon, editors, *POPL*, pages 888–901. ACM, 2017. ISBN 978-1-4503-4660-3.

Paper V :
Design and Use of PreTPost Framework

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.

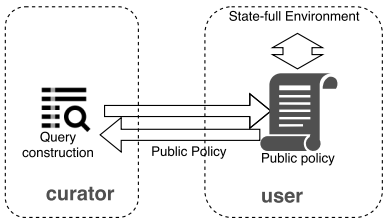

Fig. 1. Requesting the policy

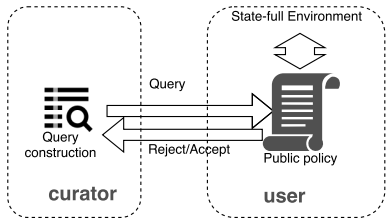

Fig. 2. Querying the policy

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.

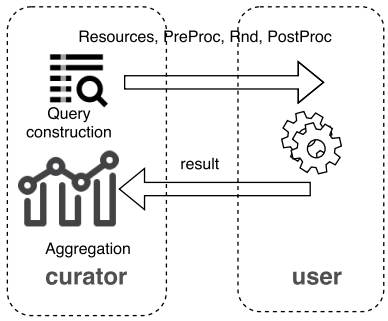

Fig. 3. Request and Response

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.

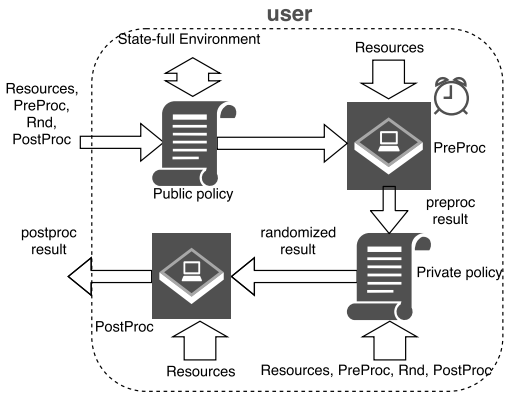

Fig. 4. Query Execution

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.