Differential Privacy: Now it's Getting Personal

Hamid Ebadi
Chalmers University of Technology
Göteborg, Sweden
hamid.ebadi@chalmers.se

David Sands
Chalmers University of Technology
Göteborg, Sweden
dave@chalmers.se

Gerardo Schneider
University of Gothenburg
Göteborg, 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, a 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.

Categories and Subject Descriptors D.3.2 [*Programming Languages*]: Language Classifications–Specialized application languages

General Terms Design, Languages, Theory

Keywords differential privacy, provenance

POPL '15, January 15 - 17 2015, Mumbai, India.

ACM 978-1-4503-3300-9/15/01…$15.00.
http://dx.doi.org/10.1145/2676726.2677005

1. Introduction

Differential privacy is a relatively new notion of privacy [5–7]. 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 [12, 17, 24]. PINQ [17] is the starting point for the present work.

PINQ-style Global Privacy Budget PINQ is an implmentation 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 2.1. *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 2.1 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 3.1 (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 3.2.

(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 3.3. *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 3.4. *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.[1] In a dy-

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

namic 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 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.

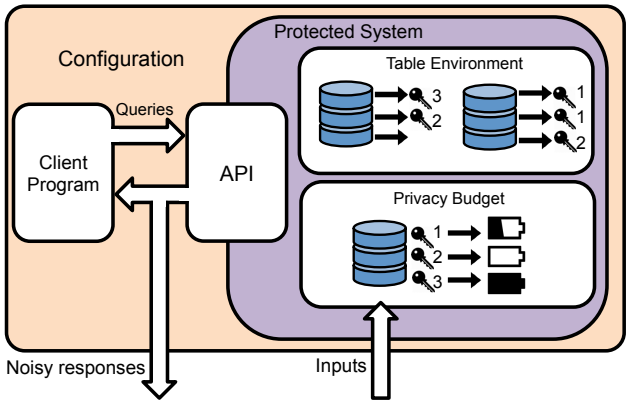

Figure 1. ProPer System Structure

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 3.4).

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 $\{\!|(r, \mathrm{Alice}), (r, \mathrm{Alice}), (r, \mathrm{Bob})|\!\}$.

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 4.1 (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 4.2. *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 4.3. *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 4.4 (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:*

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

Theorem 4.5. *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 4.6 (Provenance Table)**.** *A provenance table is a multi relation of type* $\mathrm{ProvTable} \stackrel{\text{def}}{=} \mathrm{Rec} \leftrightarrow \mathrm{Rec}_\perp$, *where* $\mathrm{Rec}_\perp \stackrel{\text{def}}{=} \mathrm{Rec} \cup \{\perp\}$ *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 4.7. *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:

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

Definition 4.8. *Given a $\uplus$-preserving function $F \in \mathrm{Table} \to \mathrm{Table}$, define $\hat{F} \in \mathrm{ProvTable} \to \mathrm{ProvTable}$ 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 4.9 (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:

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

$$\begin{aligned}
\widetilde{F} \bullet A &= \{\!|a^{[m \cdot n]} \mid a \overset{n}{\widetilde{F}} b, b \overset{m}{\in} A|\!\} \\
&= \{\!|a^{[m \cdot n]} \mid a \overset{n}{\in} F(\{b\}), b \overset{m}{\in} A|\!\} \quad \text{(Def. 4.8)} \\
&= \{\!|a^{[k]} \mid a \overset{k}{\in} F(A)|\!\} \quad (F \text{ preserves } \uplus) \\
&= F(A)
\end{aligned}$$

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:

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

□

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 4.10 (Protected System States)**.**

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

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.

Here we assume that F ranges over an unspecified set denoting $\uplus$-preserving functions in $\text{Table} \to \text{Table}$, 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).

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

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

Figure 2. ProgAct: the labels of the transition system

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 4.11 (Client Program)**.** *A client program (a program for short) is a labelled transition system $\langle \mathbb{P}, \to, P_0 \rangle$ where $\mathbb{P}$ is the set of program states with initial state P_0 and transition relation $\to \subseteq (\mathbb{P} \times \text{ProgAct} \times \mathbb{P})$, 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 $\langle \mathbb{P}, \to, P_0 \rangle$ 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*:

$$\mathbb{C} \in \text{Config} \overset{\text{def}}{=} \mathbb{P} \times (\text{TVar} \to \text{ProvTable}) \times (\text{Rec} \rightharpoonup \mathbb{R}^+)$$

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.

Definition 4.12 (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 4.13 (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 4.16. 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 4.14 (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:

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

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 4.15 (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 4.16 (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 $\mathbb{C} \xrightarrow{a}_p \mathbb{C}'$ and $\mathbb{C} \xrightarrow{a}_q \mathbb{C}''$ 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 5.1 (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 $T \stackrel{r}{\sqsubset} T'$ to various structures containing tables. In all cases we define the overloaded relation $\stackrel{r}{\sim}$ to be the symmetric closure of $\stackrel{r}{\sqsubset}$, so $T \stackrel{r}{\sim} T'$ iff either $T \stackrel{r}{\sqsubset} T'$ or $T' \stackrel{r}{\sqsubset} T$.

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

Theorem 5.2 ($\mathscr{E}$-differential privacy for all traces)**.** *If $\sigma \stackrel{r}{\sim} \sigma'$ and $\Pr(\mathbb{C} \stackrel{\sigma}{\Rightarrow}) = p$ then $\Pr(\mathbb{C} \stackrel{\sigma'}{\Rightarrow}) = q$ for some q such that $p \leq q \cdot \exp(\mathscr{E}(r))$.*

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

$$\text{Assign}\ \frac{P \xrightarrow{tv:=e} P'}{\langle P, E, B\rangle \xrightarrow{\tau}_1 \langle P', E[tv \mapsto D], B\rangle} \quad \text{where } D = \begin{cases} \hat{F}(E(tv_1) \uplus \cdots \uplus E(tv_n)) & \text{if } e = F(tv_1 \uplus \cdots \uplus tv_n) \\ \mathsf{Const}(T) & \text{if } e = T \\ E(tv') & \text{if } e = tv' \end{cases}$$

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

$$\text{Query}\ \frac{P \xrightarrow{Q_\varepsilon(tv)?n} P'}{\langle P, E, B\rangle \xrightarrow{n}_p \langle P', E, B[r \mapsto B(r) - C(r) \mid r \in A]\rangle} \quad \text{where } \begin{array}{rcl} C & = & [s \mapsto \varepsilon.\mathsf{size}(E(tv) \bullet \{s\}) \mid s \in \mathsf{dom}(B)] \\ A & = & \{r \mid B(r) \geq C(r)\} \\ L & = & E(tv) \bullet (A \cup \{\bot\}) \\ p & = & \Pr(Q_\varepsilon(L) = n) \end{array}$$

Figure 3. Operational Semantics

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

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

Proof.

$$\begin{aligned} D \overset{r}{\sqsubset} D' &\Rightarrow (D' \rhd \{r\}) \uplus D = D' \\ &\Rightarrow \mathsf{dom}((D' \rhd \{r\}) \uplus D) = \mathsf{dom}(D') \\ &\Rightarrow \mathsf{dom}(D' \rhd \{r\}) \uplus \mathsf{dom}(D) = \mathsf{dom}(D') \\ &\Leftrightarrow D' \bullet \{r\} \uplus \mathsf{dom}(D) = \mathsf{dom}(D') \qquad (4.3(iv)) \ \square \end{aligned}$$

Proposition 5.5. $\forall i \in I.(D_i \overset{r}{\sqsubset} D'_i) \Rightarrow \biguplus_{i \in I} D_i \overset{r}{\sqsubset} \biguplus_{i \in I} D'_i$

Proof.

$$\begin{aligned} &\forall i \in I.(D'_i \rhd \{r\}) \uplus D_i = D'_i \\ &\Rightarrow \biguplus_i ((D'_i \rhd \{r\}) \uplus D_i) = \biguplus_i D'_i \\ &\Rightarrow \biguplus_i (D'_i \rhd \{r\}) \uplus (\biguplus_i D_i) = \biguplus_i D'_i \\ &\Leftrightarrow (\biguplus_i D'_i \rhd \{r\}) \uplus (\biguplus_i D_i) = \biguplus_i D'_i \qquad (4.3(ii)) \\ &\Leftrightarrow \biguplus_{i \in I} D_i \overset{r}{\sqsubset} \biguplus_{i \in I} D'_i \qquad (\text{def of } \overset{r}{\sqsubset}) \qquad \square \end{aligned}$$

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

Proposition 5.6. $D \overset{r}{\sqsubset} D' \Rightarrow \hat{F}(D) \overset{r}{\sqsubset} \hat{F}(D')$

Proof. Assume premise.

$$\begin{aligned} \hat{F}(D') &= \widetilde{F} \,;\, D' && (\text{def.}) \\ &= \widetilde{F} \,;\, (D' \rhd \{r\} \uplus D) && (\text{premise}) \\ &= (\widetilde{F} \,;\, (D' \rhd \{r\})) \uplus (\widetilde{F} \,;\, D) && (4.3(i)) \\ &= ((\widetilde{F} \,;\, D') \rhd \{r\}) \uplus (\widetilde{F} \,;\, D) && (4.3(iii)) \\ &= ((\hat{F}(D')) \rhd \{r\}) \uplus \hat{F}(D) && (\hat{F} \text{ def.}) \end{aligned}$$

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

Definition 5.7 ((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 5.2 establishes the invariant relation between configurations from which the Theorem is a straightforward corollary.

Lemma 5.8. *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

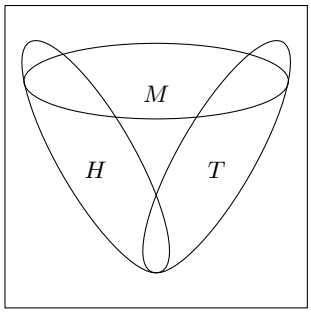

Figure 4. T=*Teachers*, M= *Managers* and H=*Headmasters*

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:

```
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. 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

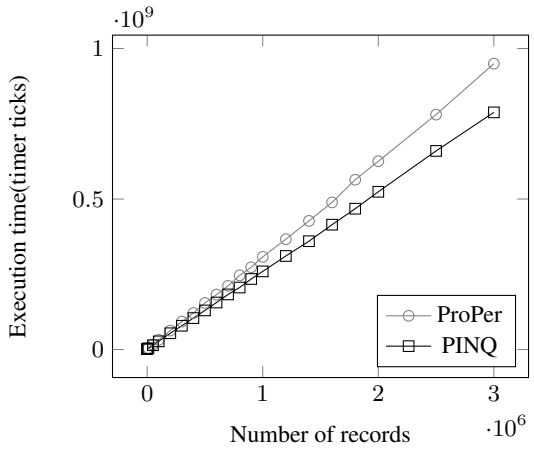

Figure 5. Comparing PINQ and ProPer wrt execution time

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 [6, 7]. 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 [10, 12, 23] 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 [20, 27]. 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 *what 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 [14, 15, 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] A. Birgisson, F. McSherry, and M. 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] J. Cheney, L. Chiticariu, and W.-C. Tan. Provenance in databases: Why, how, and where. *Found. Trends databases*, 1(4):379–474, Apr. 2009. .

[3] Y. Cui and J. 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] S. B. Davidson, S. Khanna, S. Roy, J. Stoyanovich, V. Tannen, and Y. Chen. On provenance and privacy. In *Proceedings of the 14th International Conference on Database Theory*, pages 3–10. ACM, 2011.

[5] C. 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] C. 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.

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

[8] C. Dwork, F. McSherry, K. Nissim, and A. Smith. Calibrating noise to sensitivity in private data analysis. In *Theory of Cryptography*, pages 265–284. Springer, 2006.

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

[10] M. Gaboardi, A. Haeberlen, J. Hsu, A. Narayan, and B. 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] P. W. Grefen and R. 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] A. Haeberlen, B. C. Pierce, and A. Narayan. Differential privacy under fire. In *USENIX Security Symposium*, 2011.

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

[14] G. Kellaris and S. 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] C. Li, M. Hay, V. Rastogi, G. Miklau, and A. 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] F. McSherry and R. Mahajan. Differentially-private network trace analysis. *SIGCOMM Comput. Commun. Rev.*, 40(4):123–134, 2010. ISSN 0146-4833.

[17] F. 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] D. Mir, S. Muthukrishnan, A. Nikolov, and R. 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] I. Mironov. On significance of the least significant bits for differential privacy. In *ACM Conference on Computer and Communications Security*. ACM, 2012.

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

[21] C. Palamidessi and M. 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] D. Proserpio, S. Goldberg, and F. 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] J. Reed and B. 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] I. Roy, S. T. V. Setty, A. Kilzer, V. Shmatikov, and E. Witchel. Airavat: Security and privacy for mapreduce. In *NSDI*, pages 297–312. USENIX Association, 2010.

[25] M. C. Tschantz, D. Kaynar, and A. Datta. Formal verification of differential privacy for interactive systems (extended abstract). *Electron. Notes Theor. Comput. Sci.*, 276:61–79, Sept. 2011.

[26] L. 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. T. Wittbold and D. M. Johnson. Information flow in nondeterministic systems. In *IEEE Symposium on Security and Privacy*, pages 144–161, 1990.

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

A. Proof of Main Lemma (5.8)

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 $\mathbb{C}_0 \overset{\sigma_1}{\Rightarrow}_{p_1} \langle P_1, E_1, B_1 \rangle \overset{a}{\rightarrow}_{p_2} \langle P, E, B \rangle = \mathbb{C}$, 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:

$$\mathbb{C} = \langle P, E_1[tv \mapsto \mathsf{Id}(T)], B_1[s \mapsto \mathscr{E}(s) \mid s \in T] \rangle$$

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

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

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

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

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

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

From here we argue by cases according to the rule applied for the last transition $\langle P_1, E_1, B_1 \rangle \xrightarrow{a}_{p_2} \langle P, E, B \rangle$. In every case except for the query transition we will see that $p_2 = 1$, and that $\langle P_1, E_1, B_1 \rangle \xrightarrow{a}_1 \mathbb{C}'$ 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

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

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

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 5.6 and 5.5 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. □