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.

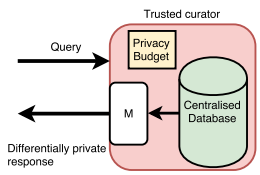

Fig. 1: Centralised Model

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.

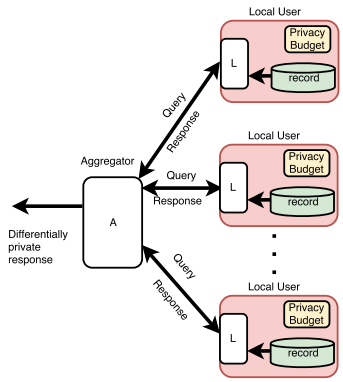

Fig. 2: Local Model

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.