Featherweight PINQ

(extended abstract)

Hamid Ebadi and David Sands

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

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

1 Introduction

Of the many of papers on differential privacy, a mere handful (at the time of writing) describe implemented systems which provide more than just a static collection of differentially private operations. The first such system is the PINQ system of McSherry [3]. PINQ is designed to allow non-experts in differential-privacy to build privacy-preserving data analyses. The system works by leveraging a fixed collection of differentially private data aggregation functions (counts, averages, etc.), and a collection of data manipulation operations, all embedded with a LINQ-like interface from otherwise arbitrary C# code. PINQ mediates all accesses to sensitive data in order to keep track of the sensitivity of various computed objects, and to ensure that the intended privacy budget is not exceeded; a budget could be exceeded by answering too many queries with too high accuracy. In this way PINQ makes sure that the analyst (programmer) does not inadvertently break differential privacy.

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

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

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

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

In this paper we provide an idealised foundation for PINQ by defining a minimalistic semantics, intended to model it's essence, while at the same time abstracting away from less relevant implementation details. In particular we model the client program completely abstractly as a deterministic labelled transition system which interacts with tables via the PINQ-like API but which is otherwise unconstrained. Our model includes all of the points above, with the exception of one: the adaptive "parallel" queries. That particular feature would require a more elaborate graph data structure to track the relationships between entities, and in the current version we only model a simpler non-adaptive parallel query.

For this model we prove differential privacy against any program.

Much of PINQ is abstracted away by our model. In fact the C# implementation of PINQ does not successfully encapsulate the all the protected parts of the system, and so some programs can violate differential privacy by bypassing the encapsulation [2], or by using side effects in places where side-effects are not intended. By idealising the interface we make clear the intended implementation, but not the details of its realisation in any particular language.

2 Model for the Idealised System

For a system to provides differential privacy it should behave similarly when the environment is initialized with two neighbouring input tables. Similar behaviour means that the probability of any outcome (trace of execution with an arbitrary size) when the system runs an arbitrary analysis is close and in the range specified by privacy budget (ε). Here the trace of a program is a sequence of internal (silent) steps and query results that are generated as a result of execution of the analysis on the database.

We use a simplified model of PINQ to prove properties we are interested in. In this section we explain our model in more detail and describe the difference between PINQ and our simplified proposed model.

Our model splits the system into a program, and the *protected system* which comprises an environment which maps program variables to tables, plus additional information needed to track stability and global budget information. The program interacts with the protected system by the following operations:

Silent operation The silent operations are introduced to mimic a class of operations that don't have any effect on the private tables or the global budget but instead change the client program state. Once a query is executed the query result is not considered private any more and computation on it is allowed outside the protected environment.

Assignment The model allows the program to manipulate a table using transformation and assign a new value to table variables. These are viewed as commands issued to the protected system. Instead of discussing about each transformation individually, we generalize it to a n-arity transformation with known stability for each argument (as described in the introduction).

Transformations Programs request transformations to be made on data by issuing a transformation command of the form $tv := F(tv_1, \ldots, tv_n)$ where F denotes a table transformer (with bounded stability), and $tv, tv_1, \ldots$ are table variables referring to tables in the environment.

Bookkeeping: Protected table & Scaling factor The environment describes, for each table variable, the table T that it represents, together with its *scaling factor*, s, which is a bound on the product of the stabilities of the transformations used to produce T.

Query A queries is the application of a (ε-) differentially private operation Q_ε on a target table. Again we do not specify a particular set of such operators.

One major difference between the model and the actual system is the way queries are executed. In the model we treat the case of a parallel query only. The standard single query being a simple instance of this.

The amount of noise that has to be added is proportional to the ε and total function sensitivity of all transformation. Scaling factor for a table represent the total function sensitivity of transformations that used to construct the table.

A query is specified with the following four arguments:

1. tv, refers to the table to be used for the analysis,
2. f, is the partitioner/key selector function that assigns/maps each record to a value in $\{1, \ldots, k\}$ for some $k \in \mathbb{N}$,
3. a vector of j ($j \leq k$) ε-DP queries $\overrightarrow{Q_\varepsilon}$, where the ith query will be applied to all records which f maps to i.

Query execution The result of execution of a query depends on available budget and the scaling factor for the table. There is enough budget if the scaling factor of the table times the cost of the queries does not exceed the global budget. If enough budget was available, a list of values $\overrightarrow{v}$ with the same size as number of queries is returned as the execution result. Otherwise $\perp$ is returned to the client program to inform it about insufficient budget. This models PINQ's budget exception. A simplification in this query model relates to sub-queries that are passed to be executed in the parallel. In PINQ each sub-query may depend on the result of another sub-query, while in this model these queries are presented to the system at the same time.

The Protected System The protected system is responsible to maintain the protected data and provide the client request with responses to queries. It also implements budgeting routines that ensures the budget consumption is under control and information leakage restrictions are enforced. The protected system can be described with following

1. a *global privacy budget* (a positive real) the overall remaining privacy budget
2. a *table environment* which maps each *table variable* to the protected table (table and scaling factor pair).

The initial state of the protected system is that some distinguished variable holds the input database of individuals, with scaling factor 1, the global privacy budget is set to the desired privacy degree, and the remaining table variables contain empty tables with scaling factor 0.

Client Program Model The probabilistic labelled transition system specified in Figure 1 defined featherweight PINQ. The model is inspired by our recent work on modelling a related system [1].

$$\begin{array}{rll}
\text{ProgAct} ::= & \tau & \text{Silent step} \\
 & \mid tv := \text{Expr} & \text{Assignment} \\
 & \mid (tv, f, \overrightarrow{Q_\varepsilon}\ ?\ \overrightarrow{v}) & \text{Primitive Queries returning } v_i \in \text{Val} \\
 & & f \in (\mathit{Tables} \to \{1, \dots, k\}) \\
 & & \overrightarrow{Q_\varepsilon} \in \prod_k Q_{\varepsilon(k)} \\
\text{Expr} ::= & tv & \text{table variable} \\
 & \mid T & \text{Table literal, } T \in \text{Table} \\
 & \mid F(tv_1, \dots, tv_n) & \text{Transform}
\end{array}$$

Fig. 1. ProgAct-labelled transition system

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

To define these transitions, we assume a client program modelled by a labelled transition system modelling the API to the protected system. For client programs, the label corresponding to a query call is of the form $(tv, f, \overrightarrow{Q_\varepsilon}\ ?\ \overrightarrow{v})$ and models the query (as described above) and the returned result as a single event. The reason for this is that it allows us to model value passing without needing to introduce any specific syntax for programs. Note that the value returned by the query is known to the program, and the program can act on it accordingly. From the perspective of the program and the protected system together, this value will be considered an observable output of the whole system.

Formally, the client program is a labelled transition system $\langle \mathbb{P}, \to, P_0 \rangle$, where $\mathbb{P}$ is all possible program states and P_0 is the initial state of the program, and the transition relation $\to \subseteq (\mathbb{P} \times \text{ProgAct} \times \mathbb{P})$ has labels ProgAct.

We assume the system is deterministic (up to results accepted from queries) and deadlock free (can always make a transition).

Configuration Semantics Each configuration is consist of program, global privacy budget and its protected system. In configuration $\langle P, E, B \rangle$, P is our program, E ranges over the protected table environments and B is a numerical value ranges over possible values for global budget. Configurations are defined as follows:

$$\langle P, E, B \rangle \in \text{Config} \overset{\text{def}}{=} \mathbb{P} \times (\text{TVar} \to (\text{Table} \times \mathbb{N})) \times \mathbb{R}^+$$

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 $a \in \text{Act} \overset{\text{def}}{=} \{\tau\} \cup \text{Val}$, and probability $p \in [0, 1]$. The definition is given by cases in Figure 2.

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

$$\text{Assign}\ \frac{P \xrightarrow{tv:=e} P'}{\langle P, E, B\rangle \xrightarrow{\tau}_1 \langle P', E[tv \mapsto (F(T_1, \ldots, T_n), s)], B\rangle}\ \text{where} \begin{cases} e = F(tv_1, \ldots, tv_n) \\ E(tv_i) = (T_i, s_i) \\ \mathit{Sensitivity}(F) = (c_1, \ldots, c_n) \\ s = \sum c_i \times s_i \end{cases}$$

$$\text{Query (out of budget)}\ \frac{P \xrightarrow{(tv, f, \overrightarrow{Q_\varepsilon}\ ?\ \perp)} P'}{\langle P, E, B\rangle \xrightarrow{\perp}_1 \langle P', E, B\rangle}\ \ \text{where } \varepsilon \cdot s > B\ \ \mathit{and}\ E(tv) = (T, s)$$

$$\text{Query}\ \frac{P \xrightarrow{(tv, f, \overrightarrow{Q_\varepsilon}\ ?\ \overrightarrow{v})} P'}{\langle P, E, B\rangle \xrightarrow{\overrightarrow{v}}_p \langle P', E, B - s \cdot \varepsilon\rangle}\ \ \text{where } \varepsilon \cdot s \geq B \text{ and } \begin{cases} E(tv) = (T, s) \\ (T_1, ..., T_n) = f(T) \\ \Pr(\overrightarrow{Q_\varepsilon}(\overrightarrow{T}) = \overrightarrow{v}) = p \end{cases}$$

Fig. 2. Operational Semantics

3 Conclusion

In this paper we presented a model for PINQ framework and other similar differentially private systems. Using the model we proved that the primitives and composition of them will not break the differential privacy guarantee promised by such systems.

References

1. Hamid Ebadi, David Sands, and Gerardo Schneider. Differential privacy: Now it's getting personal. In *Proceedings of the 42Nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '15. ACM, 2015.
2. Andreas Haeberlen, Benjamin C Pierce, and Arjun Narayan. Differential privacy under fire. In *USENIX Security Symposium*, 2011.
3. Frank D McSherry. Privacy integrated queries: an extensible platform for privacy-preserving data analysis. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*, pages 19–30. ACM, 2009.