~~Privacy Integrated Queries~~

~~An Extensible Platform for Privacy-Preserving Data Analysis~~

Frank McSherry
Microsoft Research, SVC
mcsherry@microsoft.com

ABSTRACT

We report on the design and implementation of the Privacy Integrated Queries (PINQ) platform for privacy-preserving data analysis. PINQ provides analysts with a programming interface to unscrubbed data through a SQL-like language. At the same time, the design of PINQ's analysis language and its careful implementation provide formal guarantees of differential privacy for any and all uses of the platform. PINQ's unconditional structural guarantees require no trust placed in the expertise or diligence of the analysts, substantially broadening the scope for design and deployment of privacy-preserving data analysis, especially by non-experts.

Categories and Subject Descriptors

H.3 [**Online Information Services**]: [Data Sharing]

General Terms

Algorithms, Security, Theory

Keywords

anonymization, confidentiality, differential privacy, LINQ

1. INTRODUCTION

Vast quantities of individual information are currently collected and analyzed by a broad spectrum of organizations. While these data clearly hold great potential for analysis, they are commonly collected under the premise of privacy. Careless disclosures can cause harm to the data's subjects and jeopardize future access to such sensitive information.

This has led to substantial interest in data analysis techniques with guarantees of privacy for the underlying records. Despite significant progress in the design of such algorithms, privacy results are subtle, numerous, and largely disparate. Myriad definitions, assumptions, and guarantees challenge even privacy experts to assess and adapt new techniques. Careful and diligent collaborations between non-expert data analysts and data providers is all but impossible.

This work presents a platform for interactive data analysis against live data which enforces one of the strongest known unconditional privacy guarantees: *differential privacy* [1, 2]. Differential privacy requires that computations be formally indistinguishable when run with and without any one record, almost as if each participant had opted out of the data set. The platform comprises a declarative programming language in which all written statements provide differential privacy, and an execution environment carefully implemented to respect the formal requirements of differential privacy.

The important feature of this approach is that the privacy guarantees are provided by the platform itself; they require no privacy sophistication on the part of the platform's users. This is unlike many prior instances of privacy research that rely heavily on expert design and analysis to create analyses, and expert evaluation to properly vet proposed approaches. In such a mode, non-expert analysts are unable to express themselves clearly or convincingly, and non-expert providers are unable to verify or interpret their privacy guarantees. Here the platform itself serves as a common basis for trust, even for analysts and providers with no previous experience with privacy, or even with each other.

Our advantage over prior platforms is differential privacy; its robust guarantees are compatible with many declarative operations and permit end-to-end analysis of such programs. Its guarantees hold in the presence of arbitrary prior knowledge and for arbitrary subsequent behavior, simplifying the attack model and allowing realistic, incremental deployment. Its formal nature also enables unexpected new functionality, including the use of groupby and join on sensitive attributes, the analysis of text and unstructured binary data, modular algorithm design (*i.e.* without whole-program knowledge), and analyses integrating multiple independent data sources. Perhaps most importantly, differential privacy requires no assumptions about the semantics of the underlying records; analysts will be able to write analyses directly against even the most sensitive and subtle of data sets, in their raw form, without concern of disclosure.

The main restriction of our approach is that analysts can only operate on the data from a distance: the operations are restricted to declarative transformations and aggregations; no source or derived data are ever returned to the analysts. This restriction is not entirely unfamiliar to many analysts, who are unable to personally inspect large volumes of data. Instead, they write computer programs to distill the data to manageable aggregates, on which they base further analyses. While the proposed platform introduces a stricter boundary between analyst and data, it is not an entirely new one.

1.1 An Overview of PINQ

We have implemented a prototype of our proposed architecture in a platform we call Privacy Integrated Queries. The implementation is based around C#'s LINQ language, a well-integrated declarative language extension to .NET. Data providers can use PINQ to wrap arbitrary LINQ data sources with a specified privacy allotment for each analyst. Analysts write arbitrary C# programs using PINQ data sources as if they were using unprotected LINQ data sources. PINQ's restricted language and run-time checks ensure that the provider's differential privacy requirements are respected, no matter how an analyst uses these protected data sets.

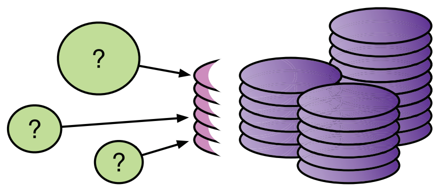

Figure 1: PINQ provides a thin protective layer in front of existing data sources, presenting an interface that appears to be that of the raw data itself.

PINQ is designed as a thin layer in front of an existing analysis engine; it does not manage data or execute queries. Instead, it supplies differentially private implementations of common transformations and aggregations written in LINQ, executed by the LINQ providers of the underlying data sets. This approach substantially simplifies our implementation, but also allows a large degree of flexibility in its deployment: a data source only needs a LINQ interface to support PINQ.

1.1.1 Mathematics of PINQ

Differential privacy requires an outcome of a computation be almost as likely with and without any one input record; we defer the quantitative form of the definition for now. Computations with this guarantee behave, from the point of view of each participant, as if their data were never used. It is a very strong requirement, in part because it makes no assumptions about prior knowledge or the data's semantics. It is also realizable; the simplest example is noisy counting: releasing of the number of records in a data set perturbed by symmetric exponential (Laplace) noise. Many other simple aggregations have similarly accurate randomized analogs.

We can significantly extend the set of differentially-private computations by introducing transformations of data sets. We provide an analysis of the "stability" of many relational transformations, showing that small changes to their inputs always result in relatively small changes to their outputs. A differentially-private analysis applied to transformed data masks small changes in the transformation's output, and so mask small changes in its inputs as well. The composed transformation and analysis will provide differential privacy. Such transformations can be composed arbitrarily, propagating differential privacy guarantees to their source data.

Finally, a sequence of differentially-private computations also provides differential privacy; the privacy depletions are at worst additive, and consequently can be tracked on-line. Analysts may pose query after query, folding their outcomes into subsequent queries, without compromising our ability to describe and constrain end-to-end privacy properties.

1.1.2 Implementation of PINQ

We have implemented PINQ as a capability-based system. PINQ allows data providers to wrap LINQ data sources in protected objects with an encoded differential privacy limit, where the objects are implemented to assess the differential privacy properties of queries and respect the imposed limit. This assessment is triggered by any aggregation, is traced through the transformations used in the query, and compared to the current limits of the participating data sources. If the assessment passes, the effective limits are decremented, and the query executed against the source LINQ provider, returning via PINQ's differentially-private implementations.

We stress that PINQ represents a very modest code base; in its current implementation it is only 613 lines of C# code. The assessment logic, following the math, is uncomplicated. The aggregations must be carefully implemented to provide differential privacy, but these are most often only a matter of post-processing the correct aggregate (*e.g.* adding noise). PINQ must also ensure that the submitted queries conform to our mathematical model for them. LINQ achieves substantial power by allowing general C# computations in predicates of `Where`, functions of `Select`, and other operations. PINQ restricts and shepherds these computations to mitigate the potential for exploitation of side channels.

1.1.3 Applications of PINQ

Programming with PINQ is done through the declarative LINQ language, in an otherwise unconstrained C# program. The analyst is not given direct access to the underlying data; instead, information is extracted via PINQ's aggregations. In exchange for this indirection, the analysis is allowed to operate on unmasked, unaltered, live records.

With a few important exceptions, programs written with PINQ look almost identical to their counterparts in LINQ. The analysts assembles an arbitrary query from permitted transformations, and specifies the accuracy for aggregations. Example 1 contains a C# PINQ fragment for counting distinct IP addresses issuing searches for an input query phrase.

Example 1 Counting searches from distinct users in PINQ.

```
var data = new PINQueryable<SearchRecord>( ... ... );

var users = from record in data
            where record.Query == argv[0]
            groupby record.IPAddress

Console.WriteLine(argv[0] + ": " + users.NoisyCount(0.1));
```

We will develop this example into a more complex search log visualization application showcasing several of PINQ's advantages over other approaches: rich data types, complex transformations, and integration into higher level applications, among many others. The full application is under one hundred lines of code and took less than a day to write.

We have written several other examples of data analyses in PINQ, including k-means clustering, perceptron classification, and contingency table measurement. These examples are relatively easy adaptations of existing approaches [3, 4]. We have also implemented association rule mining in PINQ, exhibiting many interesting trade-offs; different approaches to measuring the same quantity (in the absence of noise) can lead to strikingly different results, of incomparable quality.

1.2 Related Work

There has been a volume of research on privacy-preserving data analysis, resulting in many distinct approaches and almost as many distinct definitions of privacy. Although we can reproduce many – but not all – of these results in PINQ, its contribution is not in the existence of such reproductions, but rather it is in the manner in which they are reproduced. Each instance of prior work required substantial effort by expert researchers in design, analysis, and implementation. Those written in PINQ do not.

Several platforms for interactive data access have been proposed, ranging from simple remote access, to query auditing schemes [5], to the closely related Secure Queries [6]. Our main departure from these prior works lies in our aim of providing formal end-to-end differential privacy guarantees under arbitrary use; we are unaware of any existing analysis platform providing such guarantees.

Publication is one alternative to interactive data access, in which data sets are scrubbed, perturbed, aggregated, suppressed, and otherwise altered to mask specific information. Unfortunately, sanitization is neither easy to do, nor even to define; research in the area has yet to stabilize on robust definitions that come without known vulnerabilities [7, 8, 9]. Furthermore, these approaches intentionally contort the data before release, requiring the analyst to understand this (often intentionally secret) process before they can make valid statistical inferences. In contrast, analyses in PINQ are run against raw source data, producing results that are exactly correct up to a well defined (and small) additive error.

The cryptography community uses the phrase "privacy preserving data mining", see [10], but for cryptographically secure function evaluation, which reveals the result of the computation but no further information about the inputs other than what the result may imply. This definition sidesteps the important issue of what results are safe to release. Even simple counts accurately reported disclose information given prior knowledge or repeated use. Our concern in this paper is not only that a security breach might occur but that sensitive information may be directly disclosed even by faithful and secure execution of the computation.

Information Flow Control [11] tracks the flow of sensitive information through general computation. Such analysis has a binary description of dependence; either a computation depends on an input or it does not. Statistical analyses and aggregations are typically handled by a trusted de-classifier, using many inputs and yet treated as if it depends on none. Differential privacy appears to generalize Information Flow Control's notion of dependence to a more fine-grained and partial notion of dependence. PINQ provides a formal tool for declassification in such a context.

PINQ leans heavily on prior work on differential privacy. Many works have developed differentially-private computations, and informed the design and implementation of PINQ. Their main practical shortcoming lay in their requirement (like other work) that analysts and providers themselves establish that an implementation provides differential privacy. Furthermore, they do not provide any of the necessary tools for analysts to produce *new* differentially-private algorithms. Going forward, we would like to base our privacy guarantees on trusted components rather than reproving results from first principles, and thereby avoid the possibility of getting the algorithms, proofs, or implementations incorrect.

1.3 Contributions

PINQ's main contribution is to supply the functionality of differential privacy to its users, both analysts and providers, through tools that do not require privacy expertise to use. PINQ factors often-complex privacy reasoning into a small, transparent, and trustworthy substrate; it removes the responsibility for such reasoning from the analyst or provider, and simplifies an analysis' path from design to deployment.

Assembling this platform requires several important steps, from privacy theory, to language design, to implementation. Theoretically, we reproduce several important meta-results about properties of differential privacy and introduce the use of transformation stability reasoning to differential privacy. The language design draws substantially from LINQ, but requires careful adaptation to support differential privacy: methods like `Join` must be adapted, and new methods like `Partition` must be introduced for effective and efficient use. Finally, the implementation of a platform like PINQ has subtle issues both to support flexible and efficient deployment and to constrain exploits that indirectly leak information.

Applications written using PINQ can safely perform many computations that reasonable privacy experts have previously consider dangerous: grouping by sensitive identifiers, joining protected tables, operating on binary data or text, integrating multiple data sets, and execution without whole program knowledge, among several others. These important features have long been anathema to privacy technology, but are now available with formal privacy guarantees.

This surprising flexibility comes from differential privacy. By moving away from ad hoc and intuitional approaches to privacy we not only provide formal end-to-end guarantees, but a formal basis for expanding the space of computations. While the point of PINQ is not specifically to promote differential privacy, it is the only privacy definition we know that supports resilience to prior knowledge, secure composition, and transformation logic, among several other properties. If other privacy definitions establish the properties PINQ requires, its core ideas should lead to trusted platforms supporting their privacy guarantees as well.

As well as a useful platform in itself, we hope that PINQ's existence leads to a different approach to privacy research. Rather than conduct and publish research using ad hoc definitions, inaccessible proofs, and casual implementation, a large volume of research could derive privacy properties through common primitives with public implementations. Although much research may not be expressible with PINQ, and although PINQ is unlikely to be the final word in trustworthy privacy platforms, it demonstrates the feasibility of such a principled approach to privacy research.

1.4 Paper Outline

The paper continues in three parts, paralleling Section 1.1. We start in Section 2 by reviewing the definition of differential privacy, and presenting supporting mathematics that are necessary for end-to-end analysis of our computations. We detail PINQ's design and implementation in Section 3, as well as some advanced features and security challenges. In Section 4 we develop a sequence of applications written against PINQ to demonstrate its ease of use and generality, culminating in a visualization tool for web search queries. Finally, we conclude in Section 5 with closing comments and directions for further research.

2. MATHEMATICAL FOUNDATIONS

We now develop some supporting mathematics for PINQ. We review the privacy definition we use, differential privacy, and develop several properties necessary to expose a programmatic interface to data. Specifically, the data types we can support, common differentially-private aggregations, how several transformations of the data sets impact privacy, and how privacy guarantees of multiple analyses compose. All of our conclusions are immediate consequences of differential privacy, rather than additional assumptions or implementation details.

2.1 Differential Privacy

Differential privacy is a relatively new privacy definition, building upon the work of [1] and publicly articulated in [2]. It differs from most previous definitions in that it does not attempt to guarantee the prevention of data disclosures, privacy violations, or other bad events; instead, it guarantees that participation in the data set is not their cause.

The definition of differential privacy requires that a randomized computation yield nearly identical distributions over outcomes when executed on nearly identical input data sets. Treating the input data sets as multisets of records over an arbitrary domain and using $\oplus$ for symmetric difference:

Definition 1. *We say a randomized computation M provides ϵ-*differential privacy *if for any two data sets A and B, and any set of possible outputs $S \subseteq Range(M)$,*

$$\mathbf{Pr}[M(A) \in S] \quad \leq \quad \mathbf{Pr}[M(B) \in S] \times \exp(\epsilon \times |A \oplus B|) \ .$$

When x is much less than one, we have that $\exp(x) \approx 1 + x$. Differential privacy ensures that the behaviors of M under A and B are essentially indistinguishable when $|A \oplus B|$ is small relative to $1/\epsilon$.

The definition is not difficult to motivate to non-experts. Any potential participant can choose between two inputs to the computation M: a data set containing their records (A) and the equivalent data set with their records removed (B). Their privacy concerns stem from the belief that these two inputs may lead to noticeably different outcomes for them. However, differential privacy requires that *any* output event (S) is almost as likely to occur with these records as without. From the point of view of any participant, computations which provide differential privacy behave almost as if their records had not been included in the analysis.

Taking a concrete example, consider the sensible concern of most web search users that their name and search history might appear on the front page of the New York Times [12]. For each participant, there is some set S of outputs of M that would prompt the New York Times to this publication; we do not necessarily know what this set S of outputs is, but we need not define S for the privacy guarantees to hold. For all users, differential privacy ensures that the probability the New York Times publishes their name and search history is barely more than had it not been included as input to M. Unless the user tells someone else, this is improbable indeed.

One important distinction between differential privacy and most other definitions is that it only bounds the *change* in probability of an event S; it does not discuss the probability of the event itself. The event may be possible or even likely. Nonetheless, it is inappropriate to charge the mechanism with mishandling a participant's data if the disclosure would have been as likely to occur even without these records.

2.1.1 Data Types

Differential privacy relies only on the assumption that the data sets are comprised of records, and is most meaningful when there are few records for each participant. It requires no assumptions about the types of the underlying records. Its privacy guarantees are not a consequence of classifying attributes as sensitive or non, nor perturbing the source data, nor suppressing values that are scarce or sensitive.

Independence of data type is a very liberating property. We needn't worry about customizing privacy guarantees for different domains, misclassifying attributes as insensitive, or overlooking sensitive combinations of insensitive attributes. We can provide meaningful guarantees for unstructured data, like free text and binary data that have previously vexed sensitivity classification. We can even support mutable records, replacing each record with a timeline of its contents.

Furthermore, by ignoring entirely the records' semantics we can provide guarantees for arbitrary functions of them. This property is fundamental to allowing analysts to write their own ad-hoc analyses, rather than choose from a set of pre-screened computations over declassified attributes.

2.2 Aggregations: Noisy Counts

The simplest differentially-private aggregation (from [1]) releases the number of records in a data set, perturbed by symmetric exponential (Laplace) noise, with density function $p(x) \propto \exp(-|x|)$, as in Figure 2.

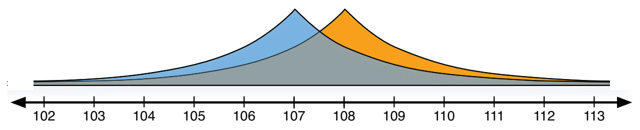

Figure 2: Adding symmetric exponential noise to counts causes the probability of any output (or set of outputs) to increase or decrease by at most a multiplicative factor when the counts are translated.

Changing an input data set from A to B can shift the true count by at most $|A \oplus B|$. The Laplace distribution is chosen because it has the property that translating its center (shifting the true value) by one unit scales the probability of any output by a multiplicative factor of at most $\exp(1)$. If the noise is first multiplied by $1/\epsilon$ this becomes $\exp(\epsilon)$, resulting in ϵ-differential privacy.

Theorem 1. *The mechanism $M(X) = |X| + Laplace(1/\epsilon)$ provides ϵ-differential privacy.*

Proof. From the definition of the Laplace distribution, for any input A the probability density of $M(A)$ at x is

$$\mathbf{Pr}[M(A) = x] \quad \propto \quad \exp(-\epsilon \times |x - |A||) \ .$$

Using the triangle inequality $|x - |A|| \geq |x - |B|| - ||A| - |B||$, and noting that $||A| - |B|| \leq |A \oplus B|$, we derive

$$\mathbf{Pr}[M(A) = x] \quad \leq \quad \mathbf{Pr}[M(B) = x] \times \exp(\epsilon \times |A \oplus B|) \ .$$

Differential privacy follows by integrating x over S. $\square$

The Laplace distribution has exponential tails in both directions, and the probability that the error exceeds t/ϵ in either direction is exponentially small in t. The released counts are very likely to be close to the true counts.

2.2.1 *Other Primitive Aggregations*

There are many other mechanisms that provide differential privacy; each paper on the subject typically contains several. To date each has privacy established as above, by written mathematical proof based on intended behavior. While this is clearly an important step in developing such a computation, the guarantees are only as convincing as the proof is accessible and the implementation is correct.

Our goal is to enable the creation of as many differentially-private computations as possible using only a few primitive components, whose mathematical properties and implementations can be publicly scrutinized and possibly verified. While we shouldn't preclude the introduction of novel primitives, they should be the exceptional, rather than default, approach to designing differentially-private algorithms.

2.3 Stable Transformations

Rather that provide access to a set of fixed aggregations, with limited potential for creative use, we intend to supply analysts with a programming language they can use to describe new and unforeseen computations. Most of the power of PINQ, and one of its main contributions, lies in arming the analyst with a rich set of transformations to apply to the data set before differentially-private aggregations.

We start by identifying a general parameter of transformations that allows us to bound the privacy implications of arbitrary sequences of such transformations.

Definition 2. *We say a transformation T is c-*stable *if for any two input data sets A and B,*

$$|T(A) \oplus T(B)| \quad \leq \quad c \times |A \oplus B| \ .$$

Transformation stability will play a central role in PINQ. Transformations with bounded stability constants propagate differential privacy guarantees made of their outputs back to their inputs, diminished by their stability constant.

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

Proof. Using the definitions of differential privacy and c-stability, we see that for any A and B,

$$\begin{aligned} & \mathbf{Pr}[M(T(A)) \in S] \\ \leq \quad & \mathbf{Pr}[M(T(B)) \in S] \times \exp(\epsilon \times |T(A) \oplus T(B)|) \\ \leq \quad & \mathbf{Pr}[M(T(B)) \in S] \times \exp(\epsilon \times c \times |A \oplus B|) \ . \end{aligned}$$

$M \circ T$ satisfies the definition of $(\epsilon \times c)$-differential privacy. □

Differentially-private aggregations applied even to multiply transformed data sets have precise privacy implications for the source data. The bounds result from repeated application of Theorem 2, compounding the stability constants of the applied transformations with the ϵ value of the analysis.

Remark. An early form of transformation can be seen in the first differentially-private algorithms for counts over arbitrary subsets of the domain. The subset of interest could be specified by the analyst, and the noised count returned. Transformations intend to separate more formally what the analyst can do freely (*e.g.* restrict the data set) from the operations that have cost (*e.g.* measuring counts with noise). This flexibility allows us to introduce new transformations, and allows the analysts to combine the transformations as they require, reflecting their interests and expertise.

2.3.1 *Stable Transformations*

We now discuss four of the transformations that PINQ supports, `Where`, `Select`, `GroupBy`, and `Join`, to see what sort of stability bounds to expect. There are many other operations that PINQ supports, drawn from LINQ, but this set is largely representative of the issues faced.

`Where` takes as input a predicate and returns the subset of the data satisfying the predicate. The stability of `Where` is one, as the addition or deletion of a source record can change the result by at most the presence of that element.

`Select` takes and applies a function mapping each source record to a new record, of a possibly different type. `Select` commonly extracts columns from a relational database, but is substantially more general. The stability of `Select` is one, as each source record results in exactly one output record.

`GroupBy` takes a function mapping records to key values, and results in a list of groups: for each observed key, the group of records that map to that key. `GroupBy` is more complicated than the previous two transformations in that the addition or deletion of an input record can *change* an output record, not simply adding or deleting it, resulting in a symmetric difference of two. However, this is the largest change that can occur, and the stability constant is two.

We stress that the output of a `GroupBy` operation is a protected list of groups of elements, rather than a list of protected groups of elements; to achieve the latter we must wait until the special `Partition` operator of Section 3.5.

`Join` takes two data sets, key selection functions for each, and returns the list of all pairs of elements whose keys match. Unrestricted, a `Join` has the ability to multiply input records, so that a single input record can influence an arbitrarily large number of output records, implying unbounded stability.

Instead, we will use a restricted form of `Join`, in which each input data set is first grouped by its join keys, and the list of groups are then joined using their group keys. The result is a compact representation of the output of the original `Join`, as each pair of groups could in principle be expanded to their full Cartesian product. Much more importantly, however, the arrangement of the output data bundles records so that we can apply stability mathematics. Each input record participates in at most one pair of groups, and as with `GroupBy` the stability constant is at most two.

This structural restriction on `Join` limits the information that can be extracted privately, by insisting that each join key result in a single record and, for example, contribute at most one to a `NoisyCount` no matter how large the group. Nonetheless, without the pre-grouping privacy bounds simply do not exist, and `Join` would not be available (and has not been, in prior work). Moreover, the enforced grouping does not interfere with many common tasks such as the use of `Join` to link unique identifiers between data sets.

Remark. `Join` introduces the practical matter that transformations may have multiple inputs, and an applied analysis reveals information about both of its sources. This is uncomplicated unless the inputs derive from common data. Even so, a single change to a data set in common induces a bounded change in each of the transformation's inputs, and a bounded change in its output (*i.e.* the stabilities add).

2.4 Composition

Any approach to privacy must address issues of composition: that several outputs may be taken together, and should still provide privacy guarantees even when subjected to joint analysis. The issue of composition underlies many of the shortcomings of current privacy guarantees. For example, Ganta *et al.* [13] show that independent k-anonymizations of intersecting data sets can leak volumes of sensitive data. Here we review two prior results on the composition properties of differential privacy, one with a slight improvement.

For a general series of analyses with ϵ_i-differential privacy, the epsilon values add, providing $(\sum_i \epsilon_i)$-differential privacy. It is not unnatural that the privacy guarantees degrade as we expose more information; the important point is that they do so in a well-controlled manner, rather than collapsing utterly as demonstrated of k-anonymity and variants in [13]. Theorem 3 presents the bounds for sequential composition.

In the special, but not uncommon, case that the analyses operate on structurally disjoint subsets of the data, the same sequence of analyses provides $(\max_i \epsilon_i)$-differential privacy. A common example of such a sequence of analyses is the GroupBy-Aggregate analysis, where each record is assured to participate in at most one aggregation. This extends to an even richer class of algorithms, on which we expand later. Theorem 4 presents the bounds for parallel composition.

2.4.1 Sequential Composition

Any sequence of computations that each provide differential privacy in isolation also provide differential privacy in sequence. Importantly, this is true not only when they are run independently, but even when subsequent computations can incorporate the outcomes of the preceding computations.

Notationally, we explicitly index each computation by the preceding outcomes, allowing them to vary arbitrarily as a function of these values. We still require each computation satisfy differential privacy with respect to their input data.

THEOREM 3. *Let M_i each provide ϵ_i-differential privacy. The sequence of $M_i(X)$ provides $(\sum_i \epsilon_i)$-differential privacy.*

PROOF. For any sequence r of outcomes $r_i \in Range(M_i)$ we write M_i^r for mechanism M_i supplied with $r_1, \ldots, r_{i-1}$. The probability of output r from the sequence of $M_i^r(A)$ is

$$\mathbf{Pr}[M(A) = r] \quad = \quad \prod_i \mathbf{Pr}[M_i^r(A) = r_i] \; .$$

Applying the definition of differential privacy for each M_i^r,

$$\begin{aligned} & \prod_i \mathbf{Pr}[M_i^r(A) = r_i] \\ \leq \quad & \prod_i \mathbf{Pr}[M_i^r(B) = r_i] \times \prod_i \exp(\epsilon_i \times |A \oplus B|) \; . \end{aligned}$$

Reconstituting the first product into $\mathbf{Pr}[M(B) = r]$ gives the definition of $(\sum_i \epsilon_i)$-differential privacy. □

Sequential composition is crucial for any privacy platform that expects to process more than one query. Privacy definitions that are not robust to sequential composition, and there are several, should be viewed with some skepticism.

REMARK. Theorem 3 has been previously observed for indistinguishability in [14], and our proof here is identical.

2.4.2 Parallel Composition

While general sequences of queries accumulate privacy costs additively, when the queries are applied to disjoint subsets of the data we can improve the bound. Specifically, if the domain of input records is partitioned into disjoint sets, independent of the actual data, and the restrictions of the input data to each part are subjected to differentially-private analysis, the ultimate privacy guarantee depends only on the worst of the guarantees of each analysis, not the sum.

THEOREM 4. *Let M_i each provide ϵ-differential privacy. Let D_i be arbitrary disjoint subsets of the input domain D. The sequence of $M_i(X \cap D_i)$ provides ϵ-differential privacy.*

PROOF. For A and B, let $A_i = A \cap D_i$ and $B_i = B \cap D_i$, and write M_i^r for mechanism M_i supplied with $r_1, \ldots, r_{i-1}$. The probability of output r from the sequence of $M_i^r(A)$ is

$$\mathbf{Pr}[M(A) = r] \quad = \quad \prod_i \mathbf{Pr}[M_i^r(A_i) = r_i] \; .$$

Applying the definition of differential privacy for each M_i^r,

$$\begin{aligned} & \prod_i \mathbf{Pr}[M_i^r(A_i) = r_i] \\ \leq \quad & \prod_i \mathbf{Pr}[M_i^r(B_i) = r_i] \times \prod_i \exp(\epsilon \times |A_i \oplus B_i|) \\ \leq \quad & \prod_i \mathbf{Pr}[M_i^r(B_i) = r_i] \times \exp(\epsilon \times |A \oplus B|) \; . \end{aligned}$$

Reassembly gives the definition of ϵ-differential privacy. □

Whereas sequential composition is critical for any functional privacy platform, parallel composition is required to extract good performance from a privacy platform. Realistic analyses require aggregates and computation on different subpopulations. Although such operations can be analysed as sequential composition, the privacy guarantee would scale with the number of subpopulations analysed. Leveraging parallel composition, the privacy costs are fixed, independent of the number of total queries, and thus permitting relatively thorough information at a modest privacy cost.

REMARK. Theorem 4 has been previously observed for the addition of noise to disjoint subpopulation counts in [1]. However, the privacy definition used (indistinguishability) introduces a factor of two in the analogous theorem, and the factors compound with each invocation of the reasoning. Our slightly altered privacy definition allows us to apply the theorem arbitrarily without increase in the bound.

2.5 A Privacy Calculus

The theorems of this section enable a rich privacy calculus, allowing us to bound the privacy implications of arbitrary sequences of arbitrary queries composed of permitted transformations and aggregations. Importantly, we can do this reasoning in an on-line fashion. Queries which arrive in sequence have their epsilon values accumulate; queries applied in parallel require us to track only the maximum.

This simplicity allows us to avoid burdening the analyst with the responsibility of correctly or completely describing the mathematical features of their query. Even for researchers familiar with the mathematics (*e.g.* the author) the reasoning process can be quite subtle and error-prone. Fortunately, it can be automated, the subject of Section 3.

3. PINQ IMPLEMENTATION

PINQ is built atop C#'s Language Integrated Queries. LINQ is a recent language extension to the .NET framework for integrating declarative access to data streams (using a language very much like SQL) into arbitrary C# programs. Central to LINQ is the `IQueryable<T>` type, a generic sequence of records of type `T`. An `IQueryable` admits transformations such as `Where`, `Select`, `GroupBy`, `Join`, and more, returning new `IQueryable` objects over possibly new types.

PINQ's implementation centers on a `PINQueryable<T>` generic type, wrapped around an underlying `IQueryable<T>`. This type supports the same methods as an `IQueryable`, but with implementations ensuring that the appropriate privacy calculations are conducted before any execution is invoked. We stress that PINQ does not supply the execution engine. Instead, a `PINQueryable` renders all of its operations to LINQ statements executed by the underlying `IQueryable`, allowing substantial flexibility in deployment.

3.1 Data Types and Control Flow

The central type in PINQ is the `PINQueryable<T>`, a protected list of objects of type `T`. The type `T` can be arbitrary. Each `PINQueryable` is comprised of an unprotected data set (an `IQueryable`), and a second new data type, a `PINQAgent`, responsible for accepting or rejecting increments to epsilon. A `PINQueryable` supports aggregations and transformations. Aggregations test the associated `PINQAgent` to confirm that the increment to epsilon is acceptable before they execute. Transformations result in new `PINQueryable` objects with a transformed data source and a new `PINQAgent`, containing transformation-appropriate logic to forward epsilon requests to the agents of its source `PINQueryable` data sets.

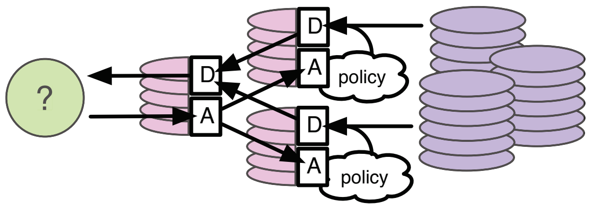

Figure 3: PINQ control/data flow. An analyst initiates a request to a PINQ object, whose agent (A) confirms, recursively, differentially-private access. Once approved by the providers' agents, data (D) flows back through trusted code ensuring the appropriate level of differential privacy.

The `PINQAgent` interface has one method, `Alert(epsilon)`, invoked before executing any differentially-private aggregation with the appropriate vale of epsilon, to confirm access. For `PINQueryable` objects wrapped around raw data sets, the `PINQAgent` is implemented by the data provider based on its privacy requirements, either from scratch or using one of several defaults (*e.g.* decrementing a per-analyst budget). For objects resulting from transformations of other `PINQueryable` data sets, PINQ constructs a `PINQAgent` which queries the `PINQAgent` objects of the transformation's inputs with transformation-appropriate scaled values of `epsilon`. These queries are be forwarded recursively, with appropriate values of epsilon, until all source data have been consulted. The process is sketched in Figure 3.

3.2 Differential Privacy Policies

Data providers dictate privacy requirements in PINQ by supplying objects that implement the `PINQAgent` interface, containing arbitrary code to mediate access to their data. Before any aggregation is performed using protected data, PINQ will invoke the `Alert` method of the associated `PINQAgent` objects with the pending privacy decrement, `epsilon`. Each `PINQAgent` is expected to respond with a boolean value, either accepting or rejecting the request.

PINQ's main role is as a privacy enforcement mechanism; it's role is not to decide who should have access to what type of data, but to enforce these decisions once they are made. Our intent is that the provider should be able to determine how much access any individual should have to a data set before they need to construct a `PINQueryable` protecting it. After the provider identifies the analyst and determines their level of access, they can then construct a simple `PINQAgent` to enforce these limits.

As examples, a privacy policy may call for different levels of access for different roles: external analysts may have a fixed budget to draw against, while internal analysts have unfettered, but logged, access. Or, if the analyst requests a less sensitive view, perhaps with certain columns stripped or groupings applied, the policy may permit finer detail and more access, in the form of more lenient `PINQAgent` objects. The decision of which constraints to apply can and should be made before the `PINQueryable` is first constructed, using a `PINQAgent` only to enforce these decisions.

Example 2 shows code to manage a fixed privacy budget.

Example 2 Implementing a fixed budget in a PINQAgent.

```
public class PINQAgentBudget : PINQAgent
{
    private double budget;

    public override bool Alert(double epsilon)
    {
        if (budget < epsilon)
            return false;

        budget = budget - epsilon;
        return true;
    }

    public PINQAgentBudget(double b)  { budget = b; }
}
```

Several other `PINQAgent` implementations exist in PINQ, ranging from enforcing a budget as above, to rate limiting information extraction by introducing artificial time delays, to simply calculating and logging the depletions of privacy. Providers are free to implement custom agents as needed, but are encouraged to perform as much privacy reasoning as possible before constructing the agent.

PINQ's use of differential privacy as a common currency provides unambiguous semantics to all of the participants, but restricts the richness of the information the `PINQAgent` has available to it at run time. This restriction is intentional. PINQ intends to be a lightweight capability-based system, and its implementation is simplified by narrowing its focus. Moreover, several operations (*e.g.* Sections 3.5 and 3.6) are based on the assumption that differential privacy is fungible: once an amount of differentially-private access is authorized, that access can be used arbitrarily, as the analyst sees fit.

3.3 Aggregation Operators

Each aggregation in PINQ takes `epsilon` as a parameter and provides ϵ-differential privacy with respect to its immediate data source. The privacy implications may be far worse for the underlying data sets from which this data set derives. Before execution, each aggregation invokes the `Alert` method of their associated `PINQAgent` with this `epsilon`, conducting the aggregation only if the eventual response is positive.

`NoisyCount` is implemented as per Theorem 1, returning the accurate `Count` of the underlying data plus Laplace noise whose magnitude is specified by the analyst, if large enough. Example 3 depicts the implementation of `NoisyCount`.

Example 3 [Abbreviated] Implementation of NoisyCount.

```
double NoisyCount(double epsilon)
{
    if (myagent.Alert(epsilon))
        return mysource.Count() + Laplace(1.0/epsilon);
    else
        throw new Exception("Access is denied");
}
```

PINQ includes other aggregations – including `NoisySum`, `NoisyAvg`, and `NoisyMed` among others – each of which takes epsilon and a function converting each record to a `double`. To provide differential privacy, the resulting values are first clamped to the interval $[-1, +1]$ before they are aggregated. This is important to ensure that a single record has only a limited impact on the aggregate, allowing a relatively small perturbation to provide differential privacy.

The implementations of these methods and the proofs of their privacy guarantees are largely prior work. `NoisySum`, like `NoisyCount`, is implemented via the addition of Laplace noise and is discussed in [1]. `NoisyMed` and `NoisyAvg` are implemented using the exponential mechanism of [15], and output values in the range $[-1, +1]$ with probabilities

$$\mathbf{Pr}[\texttt{NoisyMed}(A) = x] \quad \propto \quad \max_{\text{med}(B)=x} \exp(-\epsilon \times |A \oplus B|/2)$$
$$\mathbf{Pr}[\texttt{NoisyAvg}(A) = x] \quad \propto \quad \max_{\text{avg}(B)=x} \exp(-\epsilon \times |A \oplus B|/2)$$

Each downweights the probability of x by the fewest modifications to the input A needed to make x the correct answer.

The accuracy of `NoisyAvg` is roughly $2/\epsilon$ divided by the number of records in the data set. `NoisyMed` results in a value that partitions the input records into two sets whose sizes differ by roughly an additive $2/\epsilon$; it need not be numerically close to the actual median.

3.3.1 *Extensibility*

PINQ is intended to be an extensible platform, and permits extension via subtyping: other privacy experts can subtype the `PINQueryable` class adding differentially-private aggregations of their own, relying on PINQ's infrastructure to confirm differentially-private access. An aggregation should first query the associated `PINQAgent`, as in Example 3 above, and may then aggregate the raw data. We have implemented a restriction of the exponential mechanism [15] in this way.

At the same time, PINQ's aim is to allow analysts to write effective analyses using its tools, rather than write analyses from scratch and incorporate them as trusted methods. The `PINQueryable` is only as secure as its weakest aggregation, and new functionality should be added carefully.

3.4 Transformation Operators

PINQ's flexibility derives from its transformation operators, each of which results in a new `PINQueryable` wrapped around an updated data source. The data are not modified in LINQ, instead a new query plan is produced. The associated `PINQAgent` is wired to forward requests on to the participating source data sets before accepting, scaling `epsilon` by the transformation's stability constant.

Our implementations of many transformations are mostly a matter of constructing new `PINQueryable` and `PINQAgent` objects, wired with the appropriate parameters. Some care is taken to restrict computations, as discussed in Section 3.7. Example 4 depicts the implementation of PINQ's `GroupBy`. Most transformations require similarly simple privacy logic.

Example 4 [Abbreviated] Implementation of GroupBy.

```
PINQueryable<IGrouping<K,T>>
GroupBy<T,K>(Expression<Func<T,K>> keyFunc)
{
    // Section 3.7 explains this, and why it is needed
    keyFunc = Purify(keyFunc) as Expression<Func<T,K>>;

    // new agent with appropriate ancestor and stability
    var newagent = new PINQAgentUnary(this.agent, 2.0);

    // new data source reflecting the operation
    var newsource = this.source.GroupBy(keyFunc);

    // construct and return a new source and agent pair
    return new PINQueryable<IGrouping<K,T>>(newsource,
                                            newagent);
}
```

The `Join` transformation is our main deviation from LINQ, whose implementation pre-groups each of its input data sets by the join key before applying the unrestricted LINQ `Join`. This involves a new signature, as the reduction method takes pairs of groups of records rather than simply pairs of records, and means that LINQ programs using `Join` will need to be (slightly) rewritten to take advantage of this functionality. PINQ admits an implementation with the LINQ signatures (with pair reductions rather than pair-of-group reductions) which releases only the reduction of the first matched pair from each group. We avoided this approach to avoid the implication that such a method respects the LINQ semantics, and to force the analyst to acknowledge the departure.

3.4.1 *Extensibility*

As with aggregations, PINQ permits the careful extension of its transformations through subtyping. There are several natural methods with bounded stability that are not found in LINQ, but are nonetheless valuable. One simple example is a multi-way join, where more than two tables are joined using common keys. If implemented through repeated use of PINQ's binary `Join`, the resulting query would have stability scaling by powers of two for each application, whereas the true stability is only two for each data set. Several other similar operations exist, and subtyping allows their inclusion without requiring updates to the core PINQ libraries.

We have also subtyped the default `PINQueryable` for performance enhancements on top of the DryadLINQ cluster-based LINQ provider, in which awareness of data placement and scale permit more efficient implementations.

3.5 The Partition Operator

As indicated in Section 2.3.1, and seen again in the type signature of Example 4, the `GroupBy` operation groups the input by keys, but keeps the groups as protected records behind the privacy curtain. For many analyses, we would prefer to shatter the protected data set into multiple protected sets, along lines drawn by some user-defined key function.

Theorem 4 tells us that structurally disjoint queries cost only the maximum privacy differential, and we would like to expose this functionality to the analyst. To that end, we introduce a `Partition` operation, like `GroupBy`, but in which the analyst must explicitly provide a set of candidate keys. The analyst is rewarded with a set of `PINQueryable` objects, one for each candidate key, containing the (possibly empty) subset of records that map to the each of the associated keys. It is important that PINQ not reveal the set of keys present in the actual data, as this would violate differential privacy. For this reason, the analyst must specify the keys of interest, and PINQ must not correct them. Some subsets may be empty, and some records may not be reflected in any subset.

The `PINQAgent` objects of these new `PINQueryable` objects all reference the same source `PINQAgent`, of the source data, but following Theorem 4 will alert the agent only to changes in the maximum value of `epsilon`. The agents share a vector of their accumulated `epsilon` values since construction, and consult this vector with each update to see if the maximum has increased. If so, they forward the change in maximum. If the maximum has not increased, they accept the request.

The difference between the uses of `GroupBy` and `Partition` in PINQ can be seen in the following two queries:

Q1. How many ZIP codes contain at least 10 patients?

Q2. For each ZIP code, how many patients live there?

For Q1, a `GroupBy` by ZIP, a `Where` on the number of patients, and a `NoisyCount` gives an approximate answer to the exact number of ZIP codes with at least 10 patients. For Q2, a `Partition` by ZIP, followed by a `NoisyCount` on each part returns an approximate count for each ZIP code. As the measurements can be noisy, neither query necessarily provides a good estimate for the other. However, both are at times important questions, and PINQ is able to answer either accurately depending on how the question is posed.

The `Partition` operator can be followed not only by aggregation but by further differentially-private computation on each of the parts. It enables a powerful recursive descent programming paradigm demonstrated in Section 4.

3.6 Privacy Budgeting

An analyst using PINQ is uncertain whether any request will be accepted or rejected, and must simply hope that the underlying `PINQAgent`s accept all of their access requests. Instead, we provide a method that attempts to "allocate" a requested privacy budget, requesting access using the input budget, and returning a new `PINQueryable` with this budget hardwired into its agent (as in Example 2) if it succeeds. When the agent is destroyed, it releases any unused budget.

This budgeting operation is also useful in support of subroutines and third party code. While an analyst may pass a `PINQueryable` to any subroutine, they run the risk that the subroutine may consume all their remaining privacy budget. By pre-allocating an object with a limited privacy allotment, the analyst can ensure the appropriate insulation.

3.7 Security Issues in Implementation

Although the stability mathematics, composition properties, and definition of differential privacy provide mathematical guarantees, they do so only when PINQ is used by analysts as intended (*i.e.* the honest-but-curious model). There are numerous implementation details that must be handled properly in order to prevent mischievous analysts from opening unintended channels for information leakage. Each lends support to the argument for a concerted implementation of a well tested platform as opposed to repeated re-implementation of privacy techniques from scratch.

Many of the following issues are specific to C#, but similar issues are likely to apply to any sufficiently rich language.

3.7.1 Non-Functional Code

Methods as arguments in LINQ, such as the predicates in `Where`, functions in `Select`, and key selectors in `GroupBy` and `Join` are represented as expression trees. These data objects encode computations, which, though largely side-effect free, can invoke arbitrary C# code. Several functions are also invoked indirectly, such as `GetHashCode` and `Equals`, and can be overloaded to emit data through insecure channels. Additionally, exceptions and non-termination are another way for input methods to report on the underlying records.

Fortunately, the expression trees are relatively clear about when and where they directly invoke methods. We restrict methods to those we know [or believe] to be side-effect free, including a substantial amount of useful library code. As most C# code is not exception-free, we wrap each computation in a try-catch block that returns a default value if an exception occurs. Indirect method calls are harder to identify, but can be controlled by restricting the types permitted in the system to either base types, or anonymous types built from them. Constructors must be called explicitly, and we simply prohibit the construction of user-defined types.

A `PINQueryable` implements this checking by applying its `Purify` function to all input methods (as in Example 4). Our implementation attempts to rewrite all input methods in a believed-safe subset of the language, excepting if it fails. The method can be overridden by providers. For example, deeper code inspection could allow much more flexibility for user-defined code, but is well beyond the scope of this note.

3.7.2 Trojan IQueryables

One of the design goals of LINQ is to allow data providers to supply custom implementations of the LINQ operations on their data sets. This allows substantial flexibility in deployment, and is something PINQ would like to preserve. However, a malicious analyst could introduce an `IQueryable` whose implementation of binary operations (*e.g.* `Join`) are implemented (by the analyst) to just read the contents of the second `IQueryable` and write them out an insecure channel. Clearly, we must avoid placing trust in the implementation of any `IQueryable`, or at least delegate responsibility for this trust to the affected data providers.

To that end, binary transformations require a "handshake" between the two data sets, where the first calls the second with a request to re-invoke the initial transformation using its unprotected data source. The second can then assess the type of the caller, and the handle of the method to invoke. If trust is established – PINQ's default is to test if the types of the `IQueryable`s are identical – the second object invokes the method with its raw data, and computation continues.

4. APPLICATIONS AND EVALUATION

In this section we present data analyses written with PINQ. Clearly not all analysis tasks can be implemented in PINQ (indeed, this is the point), but we aim to convince the reader that the set is sufficiently large as to be broadly useful.

Our main example application is a data visualization based on search logs that contain IP information and query text. The application demonstrates many features of PINQ largely absent from other privacy-preserving data analysis platforms. These include direct access to unmodified data, user-supplied record-to-record transformations, operations such as `GroupBy` and `Join` on "sensitive" attributes, multiple independent data sets, and unfettered integration into higher-level programs.

Many other non-trivial analyses can be expressed in PINQ, including k-means clustering, perceptron classification, contingency table measurement, and association rule mining. Most are relatively direct, following previous research [3, 4].

For our experiments we use the DryadLINQ [16] provider. DryadLINQ is a research LINQ provider implemented on top of the Dryad [17] middleware for data parallel computation, and currently scales to at least thousands of compute nodes. Our test data sets are of limited size, roughly 100GB, and do not fully exercise the scalability of the DryadLINQ provider. We do not report on execution times, as PINQ's reasoning is an insignificant contribution, but rather the amount and nature of information we can extract from the data privately.

For clarity, we present examples written as if the data analyst is also the data provider, charged with assembling the source `PINQueryable` objects. In a real deployment, this assembly should be done on separate trusted infrastructure.

4.1 Data Analysis: Stage 1 of 3

We start with a simple application of PINQ, approximating the number of distinct search users who have searched for a specified query term. Our approach is just as in LINQ: we first transform the search records (comma-delimited strings) into tuples (string arrays) whose fields have known meaning, then restrict the data to records with the input search query, then group by the supplied IP address to get distinct users, then count the remaining records (groups of string arrays). The full program is reproduced in Example 5.

Example 5 Measuring query frequencies in PINQ.

```
// prepare data with privacy budget
var agent = new PINQAgentBudget(1.0);
var data  = new PINQueryable<string>(rawdata, agent);

// break out fields, filter by query, group by IP
var users = data.Select(line => line.Split(','))
                .Where(fields => fields[20] == args[0])
                .GroupBy(fields => fields[0]);

// output the count to the screen, or anywhere else
Console.WriteLine(args[0] + ": " + users.NoisyCount(0.1));
```

This relatively simple example demonstrates several important features of PINQ. The input data are text strings; we happen to know *a priori* that they are comma delimited, but this information plays no role in the privacy guarantees. The filtering is done against an anaylst-supplied query term, and may be frequent or infrequent, sensitive or insensitive. To get the set of distinct users we group using the logged IP address, clearly highly sensitive information.

4.2 Data Analysis: Stage 2 of 3

Our program as written gives the count for a single query, and if the analyst wants additional counts they must run the program again. This incurs additional privacy cost, and will be unsuitable for extracting large numbers of query counts.

Instead, we can rewrite the previous program to use the `Partition` operator to permit an arbitrary number of counts at fixed privacy cost. Rather than filter records with `Where`, we use the same key selection function and an input set of query strings to `Partition` the records. Having done so, we iterate through each of the queries and associated parts, grouping the records in each by IP address. To further enrich the example, we then partition each of these data sets by the number of times each has IP issued the query, before producing a noisy count. (see Example 6).

Example 6 Measuring many query frequencies in PINQ.

```
// prepare data with privacy budget
var agent = new PINQAgentBudget(1.0);
var data  = new PINQueryable<string>(rawdata, agent);

// break out fields, but partition rather than filter
var parts = data.Select(line => line.Split(','));
                .Partition(args, fields => fields[20]);

foreach (var query in args)
{
  // use the searches for query, grouped by IP address
  var users = parts[query].GroupBy(fields => fields[0]);

  // further partition by the frequency of searches
  var freqs = users.Partition(new int[] {1,2,3,4,5},
                              group => group.Count());

  // output the counts to the screen, or anywhere else
  Console.WriteLine(query + ":");
  foreach (var count in new int[] {1,2,3,4,5})
    Console.WriteLine(freqs[count].NoisyCount(0.1));
}
```

Because we use `Partition` rather than multiple `Where` calls, the privacy cost associated with the program can be seen by PINQ to be only the maximum of the privacy costs of each of the loops, exactly the same cost as in Example 5.

	Freq 1	Freq 2	Freq 3	Freq 4	Freq 5
google	356743	108336	45363	25092	14347
yahoo	140966	42379	17624	9671	5707
baidu	300	79	29	26	9
amazon	16798	3376	808	378	132
ebay	100338	26205	9564	4065	2604
cnn	25442	7492	2899	1658	919
msnbc	7828	2496	849	565	283

Table 1: Numbers of users searching for various terms, broken out by number of times they searched.

Table 1 reports measurements of a few queries on our input set. Each reported measurement is the exact count plus Laplace noise with parameter 10, corresponding to standard deviation $10\sqrt{2}$. For some measurements this error is relatively insignificant. For other measurements it is significant, but nonetheless reveals that the original value is quite small.

4.3 Data Analysis: Stage 3 of 3

We now expand out our example program from simple reporting (a not uncommon task) to a richer analysis application. Our goal is to visualize the distribution of locations of searches for various search queries. At a high level, we will transform the IP addresses into latitude-longitude pairs, by joining with a second proprietary data set, and then send the coordinates to a visualization algorithm borrowed from the work of [18]. Although we will describe the visualization algorithm at a high level, it is fundamental that PINQ provides privacy guarantees without the knowledge of what the algorithm plans to do with the data.

Starting from the prior examples, in which we have partitioned the data sets by query and grouped the results by IP address, we now demonstrate a fragment that will let us transform IP addresses into latitude-longitude coordinates. We use a second data set `iplatlon` whose entries are IP addresses and corresponding latitude-longitude coordinates. We join these two data sets, using the IP addresses in each as keys, resulting in a lat-lon coordinate pair in place of each group of searches. Example 7 contains the code for this `Join` transformation, but may take some explaining.

Example 7 Transforming IP addresses to coordinates.

```
// ... within the per-query loop, from before ...

// use the searches for query, group by IP address
var users = parts[query].GroupBy(fields => fields[0]);

// extract IP address from each group, and match
var coords = users.Join(iplatlon,
                        group => group.Key,
                        entry => entry[0],
                        (glist,elist) => elist.First());
```

The `Join` transformation in LINQ takes four parameters: the second data set, the first key selector function, the second key selector function, and finally a reduction function to apply to pairs of records. The syntax in PINQ is equivalent, except for the reduction function, which reduces a pair of groups of records. Our reduction function outputs the first lat-lon record in the second group, one per IP address.

As we can see, PINQ supports the natural and fairly common use of `Join` to transform data sets using primary keys. In such a case, we expect the groups to be singletons, and the `Join` applies as would be expected in the non-PINQ case. If we used the implementation of `Join` with LINQ's syntax we would also get the intended behavior, and in this case the PINQ and LINQ code would be literally identical.

Remark. Our second data set raises an interesting point about alternate applications of differential privacy. While the operation we perform, mapping IP addresses to latitude-longitude, is essentially just a complicated `Select`, the data set describing the mapping is proprietary. Each record in the data set required some investment of effort to produce, from which the owners presumably hope to extract value. Using this data through PINQ prevents the dissemination of individual records, preserving the value of the data set while still permitting its use. This use of proprietary data seems to have many similarities to the use of personal data. Data curation is another setting where one wants to permit access to data, but preclude dissemination of the data itself.

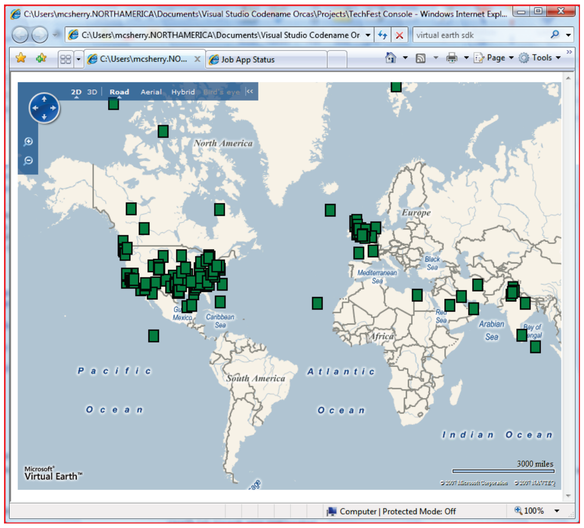

Figure 4: Example output, displaying a representative distribution of the latitude-longitude coordinates of users searching for "cricket". The computation has differential privacy not because of properties of the output itself, a quite complicated artifact, but because of the manner in which it was produced.

Finally, our algorithm takes the list of lat-lon coordinates of the IPs searching for the input search query, and invokes a `Visualization` subroutine which uses an algorithm of [18]. At a high level, this subroutine partitions the input data (geo-spatial coordinates) at increasingly fine granularities, measuring noisy counts of each region, at each granularity. From these counts, it is able to synthesize a representative distribution of data points that roughly match those trends observed in the counts; dense subregions contain many representative data points, spare regions contain relatively few. An example for the query "cricket" can be seen in Figure 4.

Readers who are not entirely sure how or why this routine works, and perhaps do not have access to [18], are in roughly the same situation as most data providers. We have almost no intuition as to why the computation should be preserve privacy, nor is any forthcoming. Nonetheless, as the routine is only provided access to the data through a `PINQueryable`, we are assured of differential privacy guarantees even without understanding the algorithm's intent or implementation. As all uses of a `PINQueryable` guarantee differential privacy, the data provider doesn't need to understand (or ever know) what the analyst plans to do with the data to be sure that differential privacy will be enforced.

Support for "modular design" of privacy algorithms is an important enabler for research and development, removing the need for end-to-end understanding of the computation. This is especially important for exploratory data analysis, where even the analysts themselves may not know the questions they will need answered until they start asking them. Removing the requirement of whole-program understanding also enables proprietary data analyses, in which an analyst may not want to divulge the analysis they intend to conduct. While the execution platform clearly must be instructed in the computations the analyst requires, the data provider does not need to be informed of their specifics.

5. CONCLUSIONS

We have presented "Privacy Integrated Queries" (PINQ), a trustworthy platform for privacy-preserving data analysis. PINQ provides private access to arbitrarily sensitive data, without requiring privacy expertise of analysts or providers. The interface and behavior are very much like that of Language Intergrated Queries (LINQ), and the privacy guarantees are the unconditional guarantees of differential privacy.

PINQ presents an opportunity to establish a more formal and transparent basis for privacy technology and research. PINQ's contribution is not only that one can write private programs, but that one can write only private programs. Algorithms built out of trusted components inherit privacy properties structurally, and do not require expert analysis and understanding to safely deploy. This expands the set of capable users of sensitive data, increases the portability of privacy-preserving algorithms across data sets and domains, and broadens the scope of the analysis of sensitive data.

5.1 Further Research Directions

The guarantees of differential privacy are rather strong, but can come at the expense of accuracy. Other weaker definitions with solid mathematical foundations do exist, but notably *approximate differential privacy* [14] also bases privacy on the differential notion of bounding change in behavior as a function of change in the input. The definition admits transformation and composition logic, and much of the PINQ infrastructure can support this definition as well.

PINQ is implemented using LINQ and inherits several of its features, including language integration, strong typing, flexible execution and optimization, and easy extensibility. The transformation stability mathematics can easily be applied to other data analysis languages, for example SQL. Other data analysis languages exist (*e.g.* scientific and statistical packages) and understanding the extent to which we can design trusted private implementations for them is open.

We have seen several example programs written in PINQ, and it is open research to consider what other analyses can be written in PINQ, and how efficiently they can use their privacy resources. Reinvestigating algorithm design with an eye towards privacy costs has the potential to inform and improve private data analysis, and new iterations of PINQ.

PINQ's extensible design allows other privacy researchers to leverage its infrastructure to support new functionality. We have seen a few examples of transformations and aggregations that are not included in PINQ, and which do not appear to be reproducible with its primitives. Expanding the set of supported operations is a natural research direction, ideally with an eye towards factoring more complex operations into commonly used primitives, minimizing the trusted base and maximizing the potential for reuse.

Acknowledgments

The author gratefully acknowledges the contributions of several collaborators. Ilya Mironov, Kobbi Nissim, and Adam Smith have each expressed substantial interest in and support for privacy tools and technology usable by non-experts. Yuan Yu, Dennis Fetterly, Úlfar Erlingsson, and Mihai Budiu helped tremendously in educating the author about LINQ, and have informed the design and implementation of PINQ. Many readers and reviewers have provided comments that have substantially improved the presentation of the paper.

6. REFERENCES

[1] C. Dwork, F. McSherry, K. Nissim, and A. Smith, "Calibrating noise to sensitivity in private data analysis," in *TCC*, 2006, pp. 265–284.

[2] C. Dwork, "Differential privacy," in *ICALP*, 2006, pp. 1–12.

[3] A. Blum, C. Dwork, F. McSherry, and K. Nissim, "Practical privacy: The SuLQ framework," in *PODS*, 2005, pp. 128–138.

[4] B. Barak, K. Chaudhuri, C. Dwork, S. Kale, F. McSherry, and K. Talwar, "Privacy, accuracy, and consistency too: a holistic solution to contingency table release," in *PODS*, 2007, pp. 273–282.

[5] N. R. Adam and J. C. Wortmann, "Security-control methods for statistical databases: A comparative study," *ACM Comput. Surv.*, vol. 21, no. 4, pp. 515–556, 1989.

[6] J. Mirkovic, "Privacy-safe nework trace sharing via secure queries," in *NDA*, 2008.

[7] P. Samarati and L. Sweeney, "Generalizing data to provide anonymity when disclosing information (abstract)," in *PODS*. ACM Press, 1998, p. 188.

[8] A. Machanavajjhala, J. Gehrke, D. Kifer, and M. Venkitasubramaniam, "l-diversity: Privacy beyond k-anonymity," in *ICDE*, 2006, p. 24.

[9] X. Xiao and Y. Tao, "M-invariance: towards privacy preserving re-publication of dynamic datasets," in *SIGMOD Conference*, 2007, pp. 689–700.

[10] Y. Lindell and B. Pinkas, "Privacy preserving data mining," in *CRYPTO*, 2000, pp. 36–54.

[11] D. E. Denning, *Cryptography and Data Security*. Addison-Wesley, 1982.

[12] M. Barbaro and T. Zeller Jr., "A face is exposed for AOL searcher no. 4417749," The New York Times, August 9, 2006.

[13] S. R. Ganta, S. P. Kasiviswanathan, and A. Smith, "Composition attacks and auxiliary information in data privacy," in *KDD*, 2008, pp. 265–273.

[14] C. Dwork, K. Kenthapadi, F. McSherry, I. Mironov, and M. Naor, "Our data, ourselves: Privacy via distributed noise generation," in *EUROCRYPT*, 2006, pp. 486–503.

[15] F. McSherry and K. Talwar, "Mechanism design via differential privacy," in *FOCS*, 2007, pp. 94–103.

[16] Y. Yu, M. Isard, D. Fetterly, M. Budiu, Úlfar Erlingsson, P. K. Gunda, and J. Currey, "DryadLINQ: A system for general-purpose distributed data-parallel computing using a high-level language," in *OSDI*, 2008.

[17] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly, "Dryad: distributed data-parallel programs from sequential building blocks," in *EuroSys*. ACM, 2007, pp. 59–72.

[18] F. McSherry and K. Talwar, "Synthetic data via differential privacy," Manuscript.