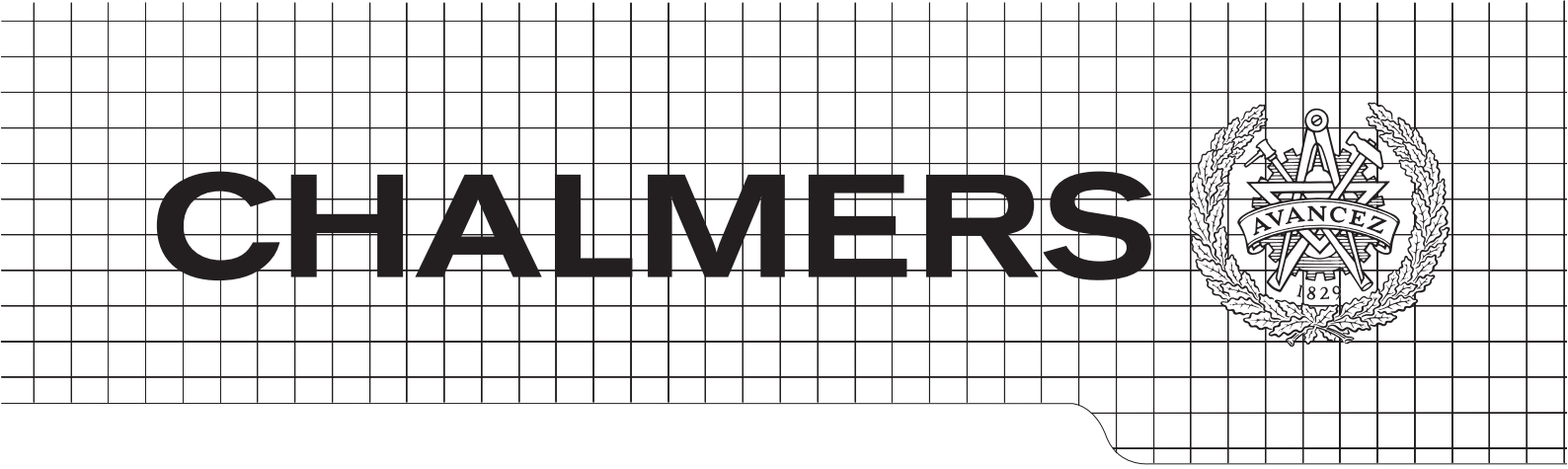

PINQuin, a framework for differentially private analysis

Master's Thesis in Computer Science

Hamid Ebadi Tavallaei

Department of Computer Science and Engineering
Secure and Dependable Computer Systems
Chalmers University of Technology
Gothenburg, Sweden 2012

PINQuin, a framework for differentially private analysis

Hamid Ebadi Tavallaei

Supervisor: Gerardo Schneider
Co-supervisor: David Sands

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

Abstract

Privacy is a humans right to seclude themselves, or information about themselves, from surveillance and view of others. Different cultures define and respect privacy differently but they usually share some basic concepts. Usually a special information that distincts one person or a group of population from others is considered personally sensitive. One aspect of privacy is related to *anonymity* which tries to remove or hide this personally sensitive information. Differential privacy is a robust standard that aims to protect individual's privacy when disclosing results from statistical analysis. In this master thesis we present a new model for differentially private data mining. As a result of this thesis we introduce a new method for privacy budgeting, namely *record based differential privacy budgeting* and PINQuin, our framework based on PINQ, that uses this new method. We also present experimental results comparing PINQuin with PINQ.

1 Introduction

There is no doubt how valuable information is in human's life. Many people eagerly pay to buy daily newspapers that only give them general information about their surrounding environment. This general information is used by most people to build a forecast and have a plan for the future. Many organizations are willing to sell information about their customers to third parties like advertising agencies. People are also willing to be influential in elections or other kinds of statistical analysis. While releasing and sharing large data sets eases large-scale data mining and helps individuals and organizations to have better data analyses and predictions, there is always the concern of individual privacy. Many anonymization operations are introduced to let data holders securely release their information, however recent de-anonymization attacks proved their inefficiency.

Differential privacy is a robust standard for data privacy proposed by Dwork based on the idea that the result of analysis should be formally indistinguishable with or without any one record [1]. Differential privacy protects individuals by providing noisy responses to statistical queries, so the result of running any arbitrary analysis on two similar databases (symmetric difference close to one) becomes indistinguishable. To achieve this privacy guarantee, access to private data is limited to sanitation functions, which are randomized algorithms, that add proper amount of noise to the results of computations.

Different platforms for privacy preserving data analysis like PINQ (Privacy Integrated Queries) [2], Fuzz [3], Airavat [4] and GUPT [5] have already been introduced. Privacy budgeting, which is common among all these platforms, is responsible for limiting information disclosure. It carefully tracks queries in order to ensure that an individual query and its aggregation with previous queries will not violate individual's privacy. Once the privacy budget for a database is consumed, the database becomes inaccessible. These frameworks deny analysts to use the database even if only one small part of the database is involved in previous queries. In this work we present a new differential privacy framework addressing some of the weaknesses of existing frameworks. In summary our contributions are:

1. A new model of privacy budgeting that tracks amount of information disclosure for each record.
2. *PINQuin*, a new framework based on PINQ, that applies this novel model.
3. Experimental results showing the feasibility of using PINQuin and its prevalence in some aspects like data utilization and flexibility in usage.

This thesis is organized as follows: Section 2 looks at the background and discusses privacy failures and attack models in general. Section 3 describes differential privacy model. Section 4 discusses about current frameworks for differential private analysis and explains how PINQ, Fuzz, Airavat and GUPT work and their utilization. In section 5, we present a new model for privacy budgeting, *Record Based Differential Privacy budgeting*, and we demonstrate our framework, PINQuin, that uses this novel model. Next, in section 6, we discuss advantages and disadvantages of this new model. Finally the last section of the report contains the conclusion.

2 Background

In this section we first explain basic concepts that are used in the privacy context like *Personally Identifiable Information (PII)* and *anonymization operations*. Next we will take a look at some well-known failures in the privacy preserving data publishing and their privacy attack models.

2.1 Personally Identifiable Information & Anonymization Operations

Different types of information from different sources are stored in a database. In the privacy context we can categorize database fields into the following four basic categories: Explicit Identifier, Quasi Identifier, Sensitive Attribute and Non Sensitive Attributes [6]. Description of each category of fields is summarized in Table 1:

Attribute name	Definition	Examples
Explicit Identifier	Explicitly identifies record owners	National identification number
Quasi Identifier	Potentially identifies record owners	ZIP code, Birth date, and Sex
Sensitive Attribute	Personal sensitive information	Score, Disease, Income
Non Sensitive Attributes	All other attributes	Favorite color

Table 1: Database fields categories

Among these fields, Personally Identifiable Information (PII) is a set of information that is used to uniquely identify a person from others. As an example in Sweden a 10 digits number (personnummer) is used as a national identification number [6].

As data usually contains personally sensitive information, it needs to be manipulated before it is made public. Anonymization is the process of hiding PII fields. This series of modifications with the goal of protecting privacy is called *anonymization operation*. There are five anonymization operations in general: Generalization, Suppression, Anatomization, Permutation and Perturbation. These operations are shortly described in Table 2.

Anonymization Operation	Definition	Example
Generalization	Replaces the value with more general value	Floor and ceiling functions
Suppression	Replaces the value with a special value in order to keep the real value private	Removing the value or replacing it with zero or a random number
Anatomization	De-associates the relationship between the quasi-identifier and sensitive information	Releasing two separate tables containing QIDs and sensitive information groups
Permutation	Shuffles values in a group	Changing the order of records
Perturbation	Replace the value with a value that keep the statistical characteristics	Adding noise

Table 2: Anonymization operations [6]

2.2 Privacy Failures

Publishing data is a big challenge for data holders. Recent failures in privacy preserving data publishing show that anonymization operations should be done more carefully. In this section we study few privacy failures that happened in the recent years.

Group Insurance Commission (GIC)

In 2000, Sweeney analyzed data from population register (public voter list) and tried to link them with patient-specific medical data through combination of birth date, zip code and gender [7]. This data was provided by GIC to researchers and also sold to industry. As some columns are common between those databases, their combination could be used to identify medical records that belong to one or a small group of people. In Figure 1 we see an example were the fields ZIP, Birthdate and Sex are common between the medical database and the voter list. In this particular case, William Weld, the governor of Massachusetts at that time and his medical records were identified in the GIC database. Sweeney showed that 87% of the U.S. population could be identified by having just a ZIP code, date of birth, and gender.

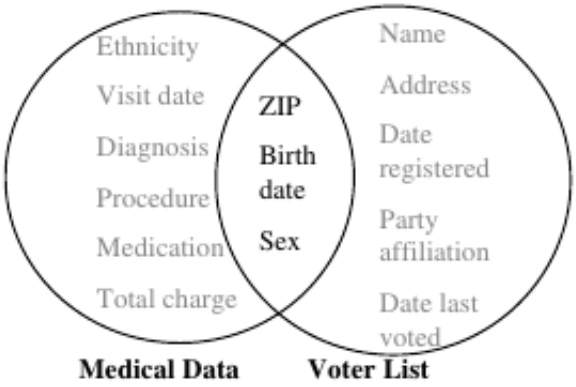

Figure 1: Linkage Attack [7]

AOL Search Log

In 2006, AOL[1] released a massive database containing around twenty million search logs related to 650000 of their users for research purpose [8]. In the database, IP addresses and user IDs were replaced with a unique random ID. This allowed researchers to see the list of keywords searched by each user. In some cases those keywords were enough to track down and identify the user. Later, AOL acknowledged the mistake and removed the data but it was too late as it was downloaded by many users and can easily be found in different web sites [8].

Netflix Prize

In 2006, Netflix[2], the world's largest online movie rental service, decided to crowd source his movie suggestion algorithm and released 100 million supposedly anonymized movie ratings. Each record included an anonymized user ID, some public information about a movie and the date on which the user rated the movie. Arvind Narayanan and Vitaly Shmatikov two researchers from the University of Texas demonstrated [9] the possibility of identifying a subscriber's record in the data set by matching their rating information with public recommendation data that can be found on the Internet Movie Database (IMDB)[3] as the background information [10]. While the data itself might seem anonymous, its combination with background information could give special meaning.

2.3 Attack Models

In this section we introduce 3 possible attack scenarios against published data, we proceed with a discussion on general problems concerning background knowledge, and finally we present alternative approaches for privacy preserving data publishing.

[1]AOL Inc. http://www.aol.com
[2]Netflix, Inc. http://www.netflix.com
[3]The Internet Movie Database (IMDB), http://www.imdb.com

In this context an *adversary* is an analyst who tries to violate a *victim*'s privacy and extract some information about him. While extracting fully accurate information from anonymized information obviously shows a problem in anonymization operation, guessing possible value with high probability is considered a privacy breach too. In the next three sections we investigate the three common possible linkage attacks.

Attack Scenarios

Record Linkage In the record linkage attacks, the adversary tries to link each record to a small number of individuals. Next, the adversary tries to use background information to narrow down the list of possible individuals and uniquely identify the victim's record. As we saw in Group Insurance Commission failure the fields: ZIP, Birthday and Sex in public voter list are considered as the Quasi Identifier (QID) and can be used to uniquely identify one individual record in the medical data. If it is not possible to single out one unique record, probability analysis on the remained records can still be used. In the probability analysis, adversary is not sure about the exact record but he may guess possible values. For example the analyst may guess the patient has either HIV or cancer. To protect against this kind of attacks *K-anonymity* guarantees there are $k-1$ other records in the table with the same Quasi Identifier. When one individual has several records in the database, k records with the same Quasi Identifier may refer to less than k record owners. For instance when one person has more than one medical record, each Quasi Identifier links to smaller group of people which increases the chance of identifying individuals. *(x,y)-anonymity* tries to solve this problem by ensuring each value of x is linked to at least k values in y.

Attribute Linkage While the previous methods like K-anonymity and (x,y)-anonymity seem to overcome the record linkage problem, if all or most linked records have the same values, adversaries can find the sensitive value without linking the owner to one unique record. For instance if all individuals in the group that are linked have the same value for an attributes, the adversary can be sure about the exact value of the attribute for the individual we are looking for, as one of the records belongs to the person. *l-diversity* tries to address this shortcoming by ensuring each group has at least "l" different values.

Table Linkage Sometimes presence or absence of one person in the database is considered a privacy breach. In this model, releasing data should not change the adversary's knowledge about the victim's personal information or his presence in the database.

Background Information

Protecting database from the described attacks, considering the background knowledge, seems to be really challenging, specially if we use a strict definition for privacy as the one described by Dalenius in 1977 [11]:

> Access to the published data should not enable the adversary to learn anything extra about target victim compared to no access to the database, even with the presence of any adversary's background knowledge obtained from other sources.

But as it was shown by Dwork this type of privacy is unachievable in a real-life application as background knowledge has a wide definition [12]. Suppose incomes are considered sensitive information and should be kept private. Assume that the statistical database discloses average income for different nationalities. An adversary who has access to this statistical information and also the auxiliary information *X's income is twice more than average* can learn X's income. One remarkable aspect of this argument is that an adversary can learn something sensitive and private about X as a result of participating in a statistical analysis. What we need is a new approach to formulate privacy so that the risk of individual privacy breach does not increase whether or not, X is in the database.

Differential privacy tries to satisfy this aspect of privacy. In differential privacy the presence or absence of any one individual should not change the analysis significantly.

Approaches for Privacy Preserving Data Publishing

There are two common approaches for privacy preserving data publishing. In the *non-interactive approach*, data publisher tries to release his huge anonymized database at once. As is plainly evident, data anonymization is not easy and really dangerous since any kind of information can be considered personal when it is combined with other background information and therefore it is not recommended to release anonymized databases.

In the *interactive model* analysts can only run a sequence of queries that each query can be constructed based on the results from previous queries. The database is responsible to track analyst activities and protect each individual's sensitive information.

Although extracting information from large data sets is needed for doing statistical analysis, anonymization is not a simple operation and highly depends on the background information. A better solution for privacy preserving data analysis is to make an interactive system instead of publishing anonymized database. In the interactive system analysts can run and see the result of queries without accessing the underlying records.

3 Differential Privacy

Differential privacy is a robust standard for data privacy proposed by Dwork based on the idea that the result of analysis should be formally indistinguishable with or without any one record [1]. This is similar to the idea that people feel safe in the population (protest) as they are indistinguishable from others.

Different platforms for privacy preserving data analysis have already been introduced, like PINQ (Privacy Integrated Queries) [2], Fuzz [3], Airavat [4] and GUPT [5]. *Privacy budgeting*, which is common in all these platforms, is responsible to limit information

disclosure. It carefully tracks queries to ensure not only an individual query but also its aggregation with other queries does not violate anyone's privacy.

This privacy standard forbids the adversary to use the database once the defined privacy budget for the database is consumed. Once this happens, the database becomes inaccessible even if only a small part of records is involved in queries. Using records from this database in any other database is considered a privacy threat, even if the record has not been used in any statistical analysis.

Differential privacy protects individuals by providing noisy responses to statistical queries, so the result of running any arbitrary analysis on two similar databases becomes indistinguishable. To achieve this privacy guarantee, access to private data is limited to sanitation functions (M), which are randomized algorithms to add proper amount of noise to the results of the performed computations.

Noise can be added to the values in records or the result of queries. It was shown in [1] that adding noise to each record for counting queries on a database of size n makes expected error on the order of $\sqrt{n}$. It is possible to have more accurate results by adding noise to the result of a query. Using this method we have constant expected error magnitude, independent of n.

We say that mechanism M preserves ϵ-differential privacy, if the probability of any specific answer does not significantly change by existence or absence of any one individual record in the data set.

Definition 1 ([2])**.** *Randomized computation M provides* ϵ-differential privacy *if for any two data sets A and B, and any set of possible outputs* $S \subset Range(M)$*,*

$$Probability[M(A) \in S] \leq Probability[M(B) \in S] \times e^{(\epsilon \times |A \ominus B|)}. \quad \square$$

The quotient $\frac{Probability[M(A) \in S]}{Probability[M(B) \in S]}$ is called *knowledge gain ratio* or *indistinguishability ratio* [13]. Differential privacy establishes that the knowledge gain ratio should be very low and close to one. Only for $\epsilon \times |A \ominus B|$ close to zero, we have that: $e^{(\epsilon \times |A \ominus B|)} \approx 1 + \epsilon \times |A \ominus B|$. There are many methods to add noise to the correct value. No matter which method is chosen, the random noise can practically be eliminated if the average of the results from arbitrary large set of queries is used. Though blocking repetitive queries may reduce privacy leakage, in more complicated attacks adversaries can issue different queries asking for the same information to make all this protective mechanism inefficient. It was shown by Sarathy and Muralidhar [13], that the number of queries rises the intruder's knowledge gain exponentially. So defining high values for ϵ has a serious impact on data privacy. As an example [14], if ϵ increases from 1 to 8.6, the knowledge gain ratio increases from 2.71 (e^1) to 5431.66 ($e^{8.6}$) .

To limit the amount of information disclosure as a consequence of asking a high number of queries, *privacy budgeting* has been introduced. Privacy budget limits the amount of information that a user can obtain about any individual, whose data is stored in the database. With privacy budgeting, multiple queries can be executed while the aggregation of these queries would not disclose more information than the defined privacy budget.

Laplacian noise is usually added to the result to make differentially private randomized computations. Its probability distribution is determined by $P(x) = \frac{1}{2b}e^{-|x-\mu|/b}$. The *mean* parameter μ is always set to zero. The *scale parameter* b is defined as $b = \frac{\Delta f}{\epsilon}$ where *query sensitivity* (Δf) equals to $\Delta f = \max \parallel f(D_1) - f(D_2) \parallel$ for all possible values D_1 and D_2. Figure 2 shows the Laplacian noise probability distribution for different values of b. When we have low values for b (high value for ϵ) it is more probable to see random noise values close to zero, as b increases (ϵ decreases) the probability of seeing a higher amount of random noise increases.

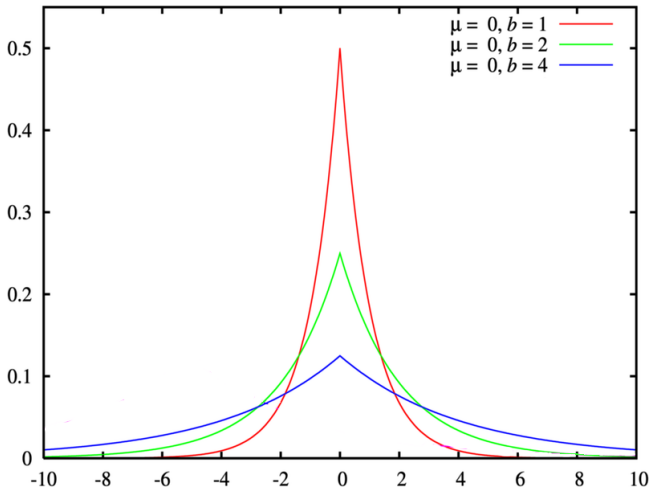

Figure 2: Laplacian noise probability distribution [15]

To adjust accuracy and change the amount of noise for different queries, we pass accuracy parameter to each aggregation method. As it is explained in the PINQ tutorial[4]:

> PINQ uses an accuracy parameter epsilon that corresponds to the privacy loss. When it is set to zero, there is no accuracy and no privacy lost either. As the value increases, the analysis becomes more and more accurate but more and more privacy is also lost.

As an example, the result from differentially private count, `NoisyCount(epsilon)`, is the result from true `Count()` plus a Laplace random variable with parameter $1.0/epsilon$. If the privacy budget is ϵ, we can independently answer k queries, where the accuracy parameter (privacy cost) for i^{th} query is ϵ_i and $\sum_{i=1}^{k} \epsilon_i \leq \epsilon$ without worrying that the aggregation of these k queries violates ϵ-differential privacy.

The system checks each new query to determine its *privacy cost* and gives execution permission only if the remaining privacy budget is sufficiently high [3]. When predefined privacy budget is consumed, it is not possible to use the database anymore. Any further

[4]Privacy Integrated Queries Tutorial, http://research.microsoft.com/en-us/projects/pinq/tutorial.aspx

access to the database violates ϵ-differential privacy, therefore it is reasonable to look for methods to run more queries and extract more information with the same privacy budget.

4 Current Frameworks

There are many frameworks that provide differential privacy for their underlying data. The main goal of these frameworks is to let normal users, with no expertise in privacy, run statistical analyses without worrying about the information leakage beyond the defined privacy policy. These frameworks are usually placed between data source and users/applications that tries to extract statistical information from the database.

In this section we present PINQ, Airavat, Fuzz and GUPT. For each framework we first give a general description and then discuss some of the problems. PINQ is presented in more detail as it is the base of our framework, PINQuin.

4.1 PINQ

Privacy Integrated Queries (PINQ) is a platform to automatically apply differential privacy principles on statistical analysis with less human expert evaluation or intervene. Empowered by LINQ declarative query language, it allows analysts to run SQL-like queries on the target databases to extract statistical information from population while protecting individual privacy.

PINQ is placed (Figure 3) as a thin layer between the query engine and the analysts. PINQ first evaluates quantitative differential privacy guarantees to ensure privacy cost of the query is within the range of privacy budget. After this phase it decreases the privacy cost from available privacy budget and passes the query to the database engine for execution. Finally proper amount of noise is added to the result returned from LINQ.

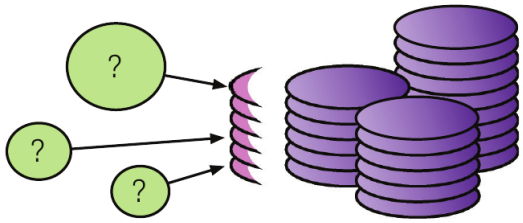

Figure 3: PINQ [2]

PINQ does not let the analyst to have access to the set of results through common database *transformation operations* like `Select, Where, Join` and `GroupBy`. Direct access to the records that are returned from these operations, endangers privacy of them. While it is not possible to see the protected records, some *aggregation functions* like `NoisyAverage(), NoisyMedian(), NoisySum()`, `NoisyOrderStatistic()` and `NoisyCount()` are provided to extract statistical information about them. These aggregation operations are also responsible to keep track of the privacy budget and add enough noise

to make analyses differentially private. Extracting information about underlying (protected) records is limited to these public aggregation methods, so transformations like `Where`, `Select` etc, should always be followed by an aggregation operation. In the next section we see how PINQ does privacy calculation for the basic database operations.

Stable transformation

To have a differentially private analysis we need to ensure one record in the input set has limited impact on the output set. Many transformations like `Where` and `Select` have limited impact on the output set as one record changes the set of output by at most one record. When applying a transformation like `Select` or `Where` on two input sets, one with a record and the other without the record, the output set changes by at most one record. If the number of changes are bounded, we have a *stable transformation*.

Definition 2. *Transformation T is c-stable if the following equation holds for any two data sets A and B* [5]*:*

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

`Select` and `Where` are two c-stable transformations and stability of both transformations are one, however some operations like `Join()` may affect the output set significantly as one record in one set can match an unbounded number of records in the other set. For instance in Figure 4 when 2 keys in the left table matches 4 keys in the right table it results in 8 new records in the resulting set For `Join()` and other transformations with unbounded stability, a restricted form, with a semantics different from the normal join operation, is defined that has bounded stability. For example, in the bounded join each record matches a bounded number of records in the other set.

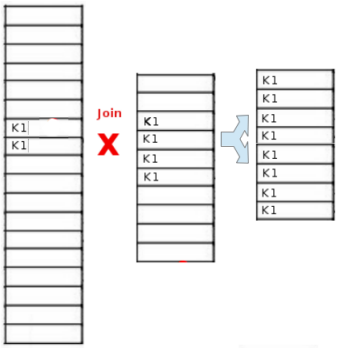

Figure 4: Join transformation

As we explained before, analysts cannot use transformations to extract information, so a randomized computation is needed to do the budgeting and adding proper amount of noise. The following relation between transformation and aggregation operations can be used for privacy calculation.

[5] $A \ominus B$ denotes the hamming distance or symmetric difference of two sets, A and B.

Theorem 1 ([2]). *Composition of a c-stable transformation with a ϵ-differential privacy computation results in $(c \times \epsilon)$-differentially private analysis .*

Aggregation

Aggregation operations take one parameter called epsilon that represents the privacy cost for a query. These aggregations usually pass epsilon to a PINQ agent object immediately. Agent objects are responsible to check if there is enough privacy budget and block the query otherwise. After this stage, the original aggregation will be executed and Laplacian noise will be applied. The amount of noise differs from one aggregation to another. Result of `NoisyCount()` is calculated by adding $Laplace(1/\epsilon)$ to the accurate `Count()`, and `NoisyAverage()` is calculated by adding $Laplace(2.0/\epsilon)$ to the actual sum of values divided by the number of records.

In the rest of the report, as seen in Figure 5, we use a curly parenthesis in figures and $R(M)$ in formulas to specify the set of records that participates in the analysis M.

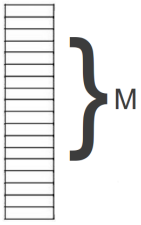

Figure 5: Running the analysis M on a subset of the records

Composition

Two types of compositions are provided, sequential and parallel. A sequence of randomized computations M_i each one providing ϵ_i-differential privacy, provides $\sum_i \epsilon_i$-differential privacy in total if they are executed sequentially. As it can be seen Figure 6, queries M_1, M_2, M_3, M_4, M_5 are executed in order. Once the privacy budget is reached the system denies answering further queries.

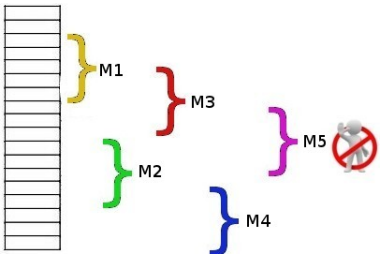

Figure 6: Sequential composition

If queries are applied to disjoint subsets of data, the privacy cost decreases and the final sequence provides $\max(\epsilon_i)$-*differential* privacy. While all queries can be executed

in sequence, parallel composition is needed for performance reasons. The same set of queries in Figure 6 can be executed in parallel. As seen in Figure 7, queries $\{M_1, M_5, M_4\}$ and also $\{M_3, M_2\}$ are applied to disjoint subset of data. The parallel execution of these queries results in less privacy budget consumption.

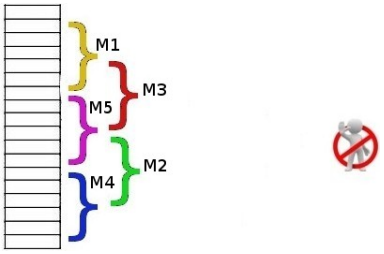

Figure 7: Parallel composition

How to use PINQ

PINQ can be considered as a wrapper around LINQ's generic `IQueryable<T>` type which stores records of type `T`. PINQ provides analysts with almost the same interfaces as LINQ but prevents the analysts to see the underlying records or results of transformations stored in the `PINQueryable`.

Program 1 shows a simple PINQ program to demonstrate simple differentially private analysis. In the first three lines of the code we initialize the database with some values of type Int and set the database budget to 10. In the rest of the code we run a simple analysis, simply counting the number of records, 11 times. The accuracy parameter for each query (`NoisyCount`) is set to 1.

Program 1 Simple differential privacy analysis in PINQ

```
var protectedData =new int[] {1,1,2,3,4,5,2,7,33,40}.AsQueryable();
var agent = new PINQAgentBudget(10.0);
var data = new PINQueryable<int>(protectedData, agent);
for (int counter=0 ; counter <11 ; counter++){
      Console.WriteLine("Noisy count: " + data.Where(num=> num<20 ).NoisyCount(1.0));
}
```

Execution of Program 1 results in 10 noisy responses, but the last query will be rejected by the system as the system runs out of budget (Listing 1).

Listing 1 Result of analysis

```
Noisy count: 8.47739016960299
Noisy count: 8.02481168051546
Noisy count: 7.8122340582543
Noisy count: 9.02969028318864
Noisy count: 7.15832354225317
Noisy count: 8.67474568106474
Noisy count: 8.77883219743362
Noisy count: 7.32924614642266
Noisy count: 8.51168064855583
Noisy count: 8.98040223671406

Unhandled Exception: System.Exception: PINQ access denied
  at PINQ.PINQueryable`1[System.Int32].NoisyCount (Double epsilon)
  [0x0004a] in PINQueryable.cs:277
  at PINQTest.MainClass.Main (System.String[] args) [0x0004c] in Main.cs:15
[ERROR] FATAL UNHANDLED EXCEPTION: System.Exception: PINQ access denied
  at PINQ.PINQueryable`1[System.Int32].NoisyCount (Double epsilon)
  [0x0004a] in PINQueryable.cs:277
  at PINQTest.MainClass.Main (System.String[] args) [0x0004c] in Main.cs:15
```

Some Problems with PINQ

Side channels in PINQ that lead to information disclosure are discussed in detail in [3]. In addition, while reviewing the code we found one missing security check in the `Distinct()` method. The method `Distinct()` has 3 arguments, the second argument is a function that takes a value of type T and returns a value of type K. While all similar functions use some sanitization mechanism to ensure the function is harmless, the `Distinct()` method does not have such security mechanism. A more secure implementation could be the one that is shown in Program 2. In this implementation the `keySelector` parameter is first passed to function `rewrite()` before execution. The function `rewrite()` is applied to all expressions that are passed from users and used to avoid exploits and side-channel attacks.[6] Lack of this check can be used to pass functions that leak information from different side channels.

4.2 Airavat

Airavat is a framework to apply differential privacy in distributed environments. Using *MapReduce model*, process of large data sets can be accelerated.

Airavat uses *Mandatory Access Control (MAC)* as the main building block to limit access to the system and prevents information leakage from common system resources

[6]The function description is found in the source code of PINQ.

Program 2 Fixing the problem with *Distinct()* method

```
public PINQueryable<T> Distinct<K>(int k, Expression<Func<T, K>> keySelector)
{
keySelector = rewrite(keySelector) as Expression<Func<T, K>>;
return NewPINQueryable<T>(source.GroupBy(keySelector).
       SelectMany(group => group.Take(k)), new PINQAgentUnary(agent, 2.0));
}
```

and other leakage channels. It runs on SELinux to enforce MAC on the distributed file system. Airavat executes trusted or untrusted (malicious) MapReduce computations on sensitive data. To apply differential privacy and protect individual records it uses a customized Java Virtual Machine and a special MapReduce framework.

As seen in Program 3 in Airavat, the *mapper* is responsible to update the budget if there is enough privacy budget, or cancel the analysis otherwise. Once the budget is updated, it maps the function to all records in the data set and clamp the value in the defined range. The *reducer* provided in Airavat takes a key and a list of values and returns the noisy result after applying the aggregation function. While mappers can be trusted or untrusted, reducers must be trusted as they are responsible to control the privacy. Reducer is usually chosen from the list that is included in the system.

Some Problems with Airavat

Airavat tries to block different side channels but there are still some low bandwidth timing and state side channels. The timing side channel can be exploited as shown in Program 4. Here procedure `process()` is executed for all records in the dataset. When the procedure finds the critical record it waits for a recognizable amount of time to signal the analyst.

Another difficulty with Airavat is its limitation in working with programs that are written based on MapReduce programming model.

4.3 Fuzz

Fuzz is another framework to analyze data while protecting individuals privacy. It is aimed to address covert channels that threat individual's privacy. PINQ and Airavat assume that analysts can observe the results of queries but other form of side channels like CPU activity, cache states, query execution time or global variables are not observable.

Fuzz isolates the physical machine and only allows analysts to communicate with the database over the network. It uses a new primitive called *predictable transaction* to close execution time side channels. Fuzz breaks each query to a set of micro-queries that each processes one record. Each micro-query is supposed to be completed within the specified time (deadline), otherwise it is canceled and a default value is used as a result of micro-query execution. If the micro-query execution takes less time, the system waits

Program 3 Mapper and Reducer in Airavat

```
\\ Mappper
if (enough privacy budget is remained)
    {
    UpdateBudget()
    foreach (Record r in Database)
        {
        Result= Map (r);
        foreach (KeyValue kv in Result)
            {
            // Range enforcment
            ClampValue(kv)
            Update(Result, kv)
            }
        emit Result
        }
}

\\ Reducer
Reduce(Key k, List val)
    {
    V= SUM( Take N element of val)
    print V + LaplaceNoise(range/epsilon)
    }
for (Length(val)-N)
    print LaplaceNoise(range/epsilon)
```

Program 4 Timing side channel

```
process(record)
    {
    if criticalRecord(record)
        Wait (x time unit) {} // or endless loop
    else
        NormalProcess()
    }
```

and returns the result after that specific time. Using this approach each query takes the same predictable amount of time for all databases of the same size.

Definition 3. *A* predictable transition *is defined as a primitive* $P-TRANS(\lambda, a, T, d)$*, where* λ *is a function,* a *is the function argument,* T *is the function timeout and* d *is the function default value. P-TRANS predictable transition takes exactly* T *time unit and returns* $\lambda(a)$ *if* λ *terminates within time* T *(with proper delay), or the systems aborts* λ *on deadline* T *and returns* d *otherwise [3].*

Fuzz introduces a well-typed functional programming environment that eases verification of programs for differential privacy properties. Any function that takes a value of a database should either return a database or return a numeric value with proper amount of noise. Sensitivity analysis and a strong type system assist Fuzz to determine the proper amount of noise and block state side channels. Fuzz introduces few critical primitives that invokes P-TRANS. Some of the primitives with their descriptions can be found in the Table 3.

Primitive	Description
map db f T d	Returns a new set containing results from applying function f to each record
split db p T d	Returns two sets, one with records that satisfy the predicate and one with records that do not
count	Add proper amount of noise to the number of records in the set
sum	Add proper amount of noise to the sum of records in the set

Table 3: Fuzz primitives

The system closes budget side channel by statistically calculating the privacy cost for each query and makes comparison with the database overall privacy budget before query execution.

Some Problems with Fuzz

While Fuzz looks like an ideal and secure framework, our practical experiments showed different results. The first shortcoming that we found was the lack of primitives to do more complicated operations like `Join()` that cannot be emulated with the current primitives.

Another important point that caught our attention was the lack of any mechanism to keep track of privacy budget. Narayan explained why this feature does not exist and how this feature can be added to the current system[7]:

> So Fuzz does not have a privacy budget, because it is implicitly assumed that a Fuzz query is going to use the entire privacy budget of the database in one

[7] Personal Communication via Email.

> shot. This is clearly a limitation. But the Fuzz implementation's primary goal was not real-world usability. It is quite clear how to implement this (in item.ml, line 94, in the definition of the function "Fuzz" we need to add an epsilon term, and do a little bookkeeping to make sure we do not exceed this limit). The goal of Fuzz was to demonstrate the protection levels required to be safe from a variety of side channel attacks.

While the previous explanation is reasonable we faced another strange behavior. After writing some programs with Fuzz we came up with Program 5 that can be used, not only to check if one person exists but also to extract her income. Program 5 maps function `highlight_one_element()` to all rows of the census database and sum up the results. Function `highlight_one_element()` first breakdowns each record to its fields. Next, if the record is the one the adversary looking for, `income * 1000000` is returned, otherwise zero is returned.

Program 5 Some problems with Fuzz

```
typedef censusline = (num,(string,(string,(string,(num,(num,(num,num)))))));

// Census db structure

function highlight_one_element (line : censusline) : num {
 let (age, aage) = line;
 let (sex, asex) = aage;
 let (race, arace) = asex;
 let (union, aunion) = arace;
 let (zipcode, azip) = aunion;
 let (income, aincome) = azip;
 if ( (sex == "F") && (zipcode == 35000) && (age == 24))
  then { income * 1000000 }  else {  0 }
}

function main (db : censusline bag) : Fuzzy string {
 sample result = Fuzz (bagsum (bagmap highlight_one_element ( 0.0) 200.0 db));

 msg =   "  income  :  " + (num2string(result)) + "\n";
 return msg
}

main
```

Running the previous query on the census database when the person described in Table 4 is in the database results in Listing 2.

ID	Age	Sex	Race	Uni	Zipcode	Income	N1	N2
9330	24	F	B	Y	35000	81164	0	1

Table 4: One sample record in census database

Listing 2 The output when the person is in the database

```
hamid@hamid:/Fuzz$ ./Fuzz queries/hamid_POC.fz -o caml-rt/query.ml   ;
cd  caml-rt/ ; make ; sudo ./runquery --mode run census.db > output.txt ;
cat output.txt ; cd .. ;

camlc -g -custom microq.o item.zo  query.zo wrapper.zo  -o runquery
  income  :  81164000000.2
5 timeout(s) total
Cycle buckets
entire program                    6470200552
Fuzz program                      6431527872
macroqueries                      6429712925
micro-query computation            901484958
micro-query waiting               5269983956
micro-query in-between              12858128
micro-query fudge wait             244142268
macroqueries unaccounted             1243615
non-macroquery                       1814947
initialization                      38672680
```

As you can see in Listing 2, the amount of income is multiplied by 1000000, but the amount of noise remained the same and the noise is so negligible that cannot hide the true value for income ($81164000000.2 \approx 81164 \times 1000000$), but when the record does not exist in the database the output will look like Listing 3. When we compare Listing 3 with Listing 2 it can be seen that the income changes from 0.0250251178281 to 81164000000.2, so existence of one specific record significantly changes the result of the query which violates differential privacy. After investigating this case in more detail, we found that the problem was in the implementation of the type system and not on the main concept that Fuzz is built on. Other frameworks (like PINQ) accepts a function to clamp (or scale) each record within the range -1 and +1 (Program 6), however this security mechanism is missing in the definition of bagsum in Fuzz.

Fuzz protects databases from covert channels at the price of longer execution time. Even though it is possible to decrease *micro-queries* (represents predictable transitions) execution time, this may result in some micro-queries not being able to finish their execution before deadline. As more and more micro-queries time out, more default values participate in the aggregation operations which decrease the accuracy of the final

Listing 3 The output when the person is not in the database

```
hamid@hamid:/Fuzz$ ./Fuzz queries/hamid_POC.fz -o caml-rt/query.ml   ;
cd  caml-rt/ ; make ; sudo ./runquery --mode run census.db > output.txt ;
cat output.txt ; cd .. ;

camlc -g -custom microq.o item.zo  query.zo wrapper.zo  -o runquery
  income  :  0.0250251178281
6 timeout(s) total
Cycle buckets
entire program                    6464987097
Fuzz program                      6431525304
macroqueries                      6429712403
micro-query computation            900053584
micro-query waiting               5347496125
micro-query in-between              11708218
micro-query fudge wait             169080328
macroqueries unaccounted            1374148
non-macroquery                      1812901
initialization                     33461793
```

Program 6 clamping the value within the range [-1,1]

```
value = scale_function(record);
if (value > 1)
    return 1;
else if (value < -1 )
    return -1;
else
    return value;
```

result. It is advised to choose the timeouts high enough so that all micro-queries can complete their execution.

4.4 GUPT

GUPT is another platform for privacy preserving data analysis that enables analysts to execute common statistical analyses while providing utility and privacy at the same time. An important feature of GUPT is its simplified definition of privacy budget in terms of accuracy. It uses a model that degrades the sensitivity of data over time and tries to automatically increase accuracy of analysis.

To reduce the amount of noise and have more accurate outputs, GUPT extends a novel framework named *"sample and aggregate"* , it first partitions the database into

smaller chunks with default size equals to $l = n^{0.4}$ (these smaller partitions are labeled as $T_1, T_2, \ldots T_l$ in figure 8) Next, a computation will be executed on these subsets and the results are clamped into the value range. Finally a customized average function adds proper noise to make the final result differentially private. Aging model empowers GUPT to find the optimal partition size that leads to more accurate results.

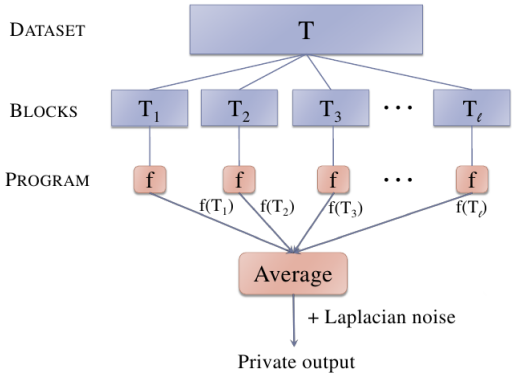

Figure 8: Sample and aggregate [5]

GUPT uses AppArmor as Mandatory Access Control (MAC) to isolate different computations and block possible state side channel attacks. To protect the system against timing attacks, GUPT sets a bound on the execution time of computations on each block and returns a default value in case one computation takes longer than the expected time.

Some problems with GUPT

We have not found any problem with this framework, so GUPT seems to be more robust than the other similar frameworks.

5 Shortcomings of Current Frameworks

Privacy budget in current differential privacy frameworks (PINQ, Fuzz, Airavat and GUPT) is defined for all records. With each query an accuracy parameter is defined that decreases the privacy balance for the whole data set even if it involves a small subset of records.

To investigate the problem in more detail let us assume a privacy limit of 1 that is imposed for the census database of personal records in which we keep every person's ID, age, income, zip-code, sex, etc. Assume the analyst is interested to have the result for the sequence of queries described in Listing 4.

The framework that uses the classic model[8] of budgeting can answer the first two queries as the sequential composition of the first two analyses does not exceed our privacy

[8]We will refer to the model that is implemented by PINQ as a *classic model*.

Listing 4 Three queries

Q_1= AVERAGE of incomes for the females older than 50 years old (privacy cost=0.3)
Q_2= SUM of incomes for the females younger than 20 years old (privacy cost=0.7)
Q_3= AVERAGE of incomes for the males older than 35 years old (privacy cost=0.5)

budget. As explained before when the system reaches its privacy budget limit, no further queries can be answered. For instance, while no question about males have been asked, the system refuses to answer any other queries, no matter whether it is about males or not. The problem arises from the fact that the database does not track information disclosure of each record (and the amount of exposure) while responding to queries.

As an improvement, PINQ [2] suggested that if queries are applied to disjoint subsets of data, parallel compositions can be used. In parallel composition, instead of summing all privacy cost, maximum privacy cost of all analyses can be used as total privacy consumption.

Figure 9: Parallel composition on disjoint subsets
Privacy budget = 1

Even though parallel composition improves database utilization, it is not always applicable. To save more privacy budget, it is necessary to find the best possible way of executing queries in parallel. In Figure 9, a parallel composition to answer all three queries (Q_1, Q_2, Q_3) with accuracy parameter equals to 0.7 can easily be found. However finding the best parallel composition for running queries requires all queries to be present in the system at the same time. This means no dependency between queries is allowed. Parallel composition is a complicated composition and in practice has some shortcomings in answering specific queries. For simple queries like the above (that deal with disjoint subsets of the database) it is possible to use parallel composition, but for more complex queries it is not always possible to simply find the disjoint subsets specially when we are dealing with different privacy values. Assuming that we run the queries in parallel (epsilon=0.7), it is clear that we lost the chance to run some more queries (Sum/Average/Count) on elements that are not the part of previous queries (like males younger than 35 or females that are between 20 and 50). So there is usually some information that can be extracted but is wasted because of the current inefficient privacy

budgeting method.

Another issue with databases that use classic differential privacy is their short lifetime. Once the budget is used there is no way to use the database, even if old records are removed and the database is gradually refilled with new records.

This is more sensible when we run statistical queries on actively changing databases like a queue that temporally holds a stream of data. For instance a network buffer where its old data is periodically replaced with new one.

6 New Approaches For Privacy Budgeting

As discussed before, some parts of data remain untouched or less used than the other parts. To improve data utilization and to extract more information from database, we propose the following two approaches, using *reference bit* and the *record based privacy budgeting*. We briefly explain both approaches in what follows.

6.1 Reference Bit

In this method, a single bit is used as the reference bit to detect untouched records. Setting a reference bit for a record shows it has been once included in an aggregation operation. For instance in the example of the queries shown in Listing 4, the execution of the first query Q_1 results in setting the reference bits for the records belong to the females older than 50:

Q_1= AVERAGE of incomes for the females older than 50 years old (privacy cost=0.3)

Later when we run Q_2 and Q_3, and consume all remained privacy budget we can still run more queries on records that are not yet referenced. So it is always possible to partition data set into two subsets, one which has already consumed all its privacy budget and the other which is remained untouched.

6.2 Record Based Privacy Budgeting

We propose to assign a privacy budget to every single record. The privacy balance increases by query accuracy when a record (i) is involved in the result of a query:

$$Balance_i = Balance_i + \epsilon.$$

Instead of starting from scratch showing that differential privacy holds for this model, we informally argue that this model provides the same privacy guarantee as the classic model.

The amount of information that is disclosed about each record in a ϵ-private mechanism does not depend on the other records or database size. Information exposure only depends on the accuracy of the randomized computation (ϵ) and is bounded by e^{ϵ} [13]. Consequently, if mechanisms M_i, each providing ϵ_i-differential privacy, are applied to the record α, the knowledge gain about α, increases to at most $e^{\sum_i \epsilon_i}$. Additional knowledge

about any other record or a group of records (auxiliary information) does not endanger that specific record. Having the exact value for all other involved records in the analysis does not help the adversary to have more information about the value of the record in question. Similarly this property holds for every other record. Whenever the privacy budget for a record is reached ($Balance_i > Budget_i$), the system stops disclosing information about the record by ignoring it and not including it in further analyses.

In the following sections, we will only focus on record based privacy budgeting which is more general, more challenging and has better performance.

7 PINQuin

PINQuin is the platform we designed to demonstrate record based privacy budgeting. PINQuin is based on PINQ and follows the same principles and syntax as PINQ. It tries to be faithful to PINQ, so with small changes PINQ can be replaced by PINQuin and programs that are written for PINQ can be converted and still be used.

In PINQuin, a privacy budget and a privacy balance are assigned to each record. PINQuin tracks how a record flows between transformations and when an aggregation operation is executed, the privacy balance for every involved record increases. Once the privacy balance for a record reaches its privacy budget, that specific record is not allowed to be used in any further analysis.

PINQuin is based on the fact that records are not related to each other and knowledge about existence of one record does not disclose any information about the others. This means PINQuin, like PINQ, does not protect group of records. Since the existence of each record is protected by differential privacy, its nonexistence or its disappearance would not disclose any information about the record.

7.1 Features

PINQuin keeps track of privacy for each record, so while PINQ can only be used for static databases, PINQuin can be used in environments where data is added, updated and removed frequently. It also lets analysts run some special set of queries that was not possible to be answered in the classic differential privacy. A big downside of PINQuin is that if the analyst is not careful about privacy budget consumption, the output will become too inaccurate to be considered useful.

7.2 How to use PINQuin

As PINQuin is based on PINQ, it shares code base and many feature and provides the analyst with almost the same programming interface as PINQ, therefore a program that is written for PINQ can be converted to its PINQuin equivalent easily. In the following sections we will discuss about PINQuin internal design and how to use its public methods[9] to analyze data in differentially private manner.

[9] More detailed information can be found in the appendix.

Internal Design

As we explained before, PINQ works just as a wrapper around LINQ's generic `IQueryable<T>` type which stores records of type T. In PINQuin for every record a key is assigned, this key is paired with the actual record of type T and the pair is added to *KeyRecord* list. There is another data structure consisting of key and budget/values to keep track of privacy budget for each record. When any record is used in an aggregation operation its corresponding budget is updated in *KeyBudgetBalance* dictionary.

A key is assigned to each record, The root PINQuin stores the actual set of records with one unique key for each record. Transformation operations are used to create a new PINQuin with a new set of records. In the resulting PINQuin, a record may influence only one record as in `Select` and `Where` operations. The `Select` transformation as can be seen in Figure 10 divides each value in the dataset by 2 and creates a new dataset with the new values and their corresponding keys. The keys are stored with the values to track information flow among transformations.

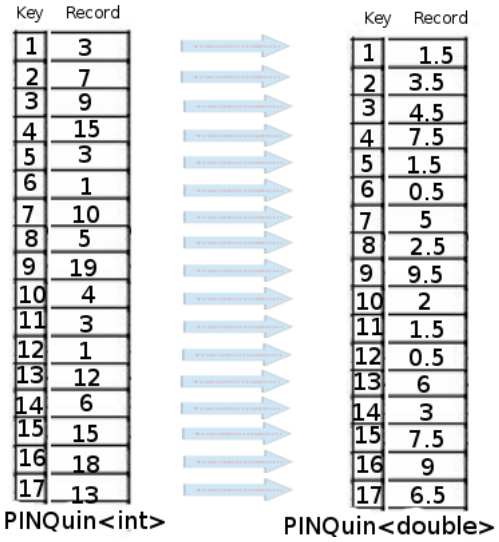

Figure 10: Each record is the result of transforming one record `Select(x => x/2)`

The `SelectMany` operation projects each element of a sequence to an `IEnumerable<T>` and flattens the resulting sequences into one sequence[10]. So a record may influence more than one record. In Figure 11 applying transformation on each record results in two records with the same keys but different values. We keep track of where the new records originates by storing the key of the source record in all the new records. In our example the first two records in the right set come from the record with the key equal to 1.

[10] http://msdn.microsoft.com/en-us/library/system.linq.enumerable.selectmany.aspx

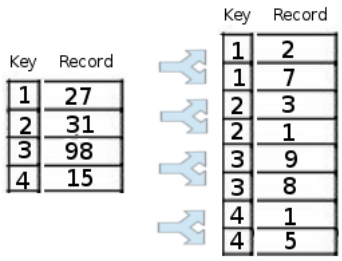

Figure 11: Each record influences many records
`SelecyMany()`

Finally, there are also cases when a record *is influenced by* more than one record. In these cases, the keys for records that have an influence on the result of the final record are stored in a list and transferred to the new PINQuin and assigned to the resulting record. As you can see in Figure 12 multiple records can contribute to create one record. In such cases list of keys that have effect on the resulting record is stored together with the record. In Figure 12 the first two records (with keys equal to 1 and 2) creates a new record that has two keys (1 and 2).

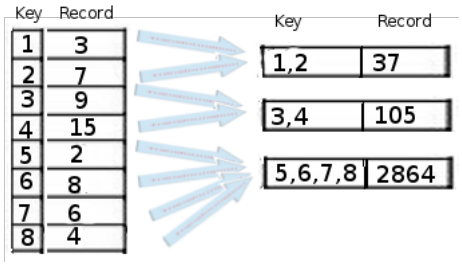

Figure 12: One record may be influenced by more than one record
`GroupBy()`

After all transformations and when it comes to the aggregation, the system checks to make sure all records that participate in the aggregation have enough privacy budget. If enough privacy budget is available, record balances are updated, otherwise those records that run out of budget cannot participate in the aggregation. Unlike PINQ, which only works as a proxy to redirect queries to the protected `IQueryable`, PINQuin keeps track of privacy for each record. PINQuin has its own implementation for all aggregation and transformation operations. Methods that are used for data initialization and data manipulation like `Insert`, `Update`, `Delete` are considered administrative methods and should be protected.

8 Pros and Cons of Record Based Privacy Budgeting

There are many advantages and disadvantages in using record based privacy budgeting. In this section we investigate pros and cons of this new model.

8.1 Advantages

In this section we discuss possible advantages of using record based differential privacy instead of the classic model.

Accuracy of Results

PINQuin can accurately respond to any sequence of queries as long as the remaining privacy budgets for all records involved in aggregation queries are higher than the privacy cost of the query. Running the same queries in PINQuin and PINQ leads to the same results till the total database privacy budget defined for PINQ data source is consumed. At this point PINQ stops responding to new queries while PINQuin can respond to more queries. After this phase, it is possible to have results with less accuracy. Inaccuracy of results depends on the number of records that run out of privacy budget. This is a responsibility of system designers and analysts to make sure this situation does not happen or the amount of noise does not affect needed accuracy. The new privacy budgeting method protects the database from privacy budget side channel attack by not involving records that have lost their privacy budget. This reduces the accuracy of the result over time but avoid privacy side channel, introduced when the system blocks a query.

Improved Utility

Data stored in a database is the most valuable part of IT systems. Collecting this information is time consuming, complicated, and an expensive task that is often subject to restrictions and strict regulations. It is extremely important to extract as much information as possible from these databases. Using this method of privacy budgeting more information can be extracted from a database while the same level of privacy is provided.

Personalized Privacy

Since privacy budget is assigned to each record, it is possible to have records with different privacy sensitivities. One person can choose to disclose more information about himself without endangering others. This can be useful specially when we are dealing with data from different sources and different sensitivities. In addition it is more flexible as it provides the possibility to update privacy budget for individuals if the record owner decides to.

Dynamic Database

This approach helps us have dynamic databases where records can be removed, added or edited without worrying about their privacy impacts. For each record that is added to the database a limited amount of privacy budget is defined.

Operations that deal with data manipulation, should be limited to database moderators and we assume the moderators are aware of the dangers that may threat individual as the result of multiple insertion of one individual. Inserting and removing each record should be done carefully to avoid reinserting any record into the database. This conveys that once a record is removed (or runs out of budget) it should not be added later.

Working on Subsets

Tracking privacy budget for each record lets us run one query on random/arbitrary chosen part of a database. If subsets have almost the same properties as the whole database, statistical analyses usually holds, while only a small part of the database consumes its privacy budget.

Protection Against Budget Side Channel

We claim that this model eliminates the privacy budget side channel introduced in [3]. In this model all queries will be answered but the accuracy decreases as more records consume all their privacy budget. So it is not possible to learn about records by observing if the query is blocked or answered.

Flexible Queries and Simplicity

The new model can answer sequence of queries that cannot be answered in the classic model. As an example, assume we need to respond queries, $Q_1, Q_2, ..., Q_n$.

For all i, $R(Q_i)$[11] has some common records with $R(Q_{i+1})$, also the set of involved records for the last query Q_n, has some common records with records from the first query, $R(Q_1)$; but there is no other common record between all these sets as it is formulated below:

$$\forall i,j; 1 \leq i < j < n; Q_i \cap Q_j \neq \emptyset \Rightarrow j = (i+1) \mod n.$$

To simplify this case, assume the relation shown in Listing 5 and Figure 13 holds for the three queries Q_1, Q_2, Q_3.

[11] We use $R(Q_i)$ to specify the set of records actively participating in the query Q_i.

Listing 5 Relation between involving records

Privacy Budget for the database = 1
Accuracy parameter for $Q_1, Q_2, Q_3 = 0.5$

$R(Q_1) \cap R(Q_2) \neq \emptyset$,
$R(Q_2) \cap R(Q_3) \neq \emptyset$,
$R(Q_1) \cap R(Q_3) \neq \emptyset$,
$R(Q_1) \cap R(Q_2) \cap R(Q_3) = \emptyset$

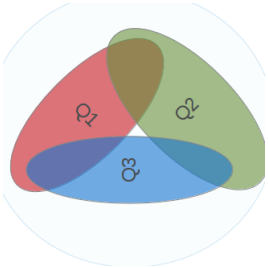

Figure 13: No intersection between the sets

As it is clear, there is no parallel composition that can split our data into disjoint subsets. Therefore in the classic model at most two out of three queries can be answered, while in the new model all three queries can be answered.

As a more realistic example, assume that the involved records for all queries have some common items. This setting is specified in Listing 6 and Figure 14. As before,

Listing 6 Relation between involving records

Q_1 = AVERAGE age of teachers.
Q_2 = AVERAGE salary of dentists.
Q_3 = COUNT of football players.

Privacy Budget for the database = 1
Accuracy parameter for, $Q_1, Q_2, Q_3 = 0.5$
$R(Q_1) \cap R(Q_2) \cap R(Q_3) \neq \emptyset$

in the classic model only two out of three queries can be answered. Since the privacy balance for the database exceeds the budget, the last query will be rejected.

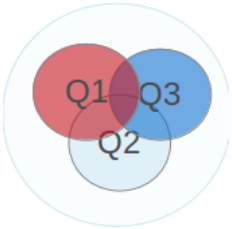

Figure 14: The parallel composition weakness

In our model, common records between all queries $R(Q_1) \cap R(Q_2) \cap R(Q_3)$ are ignored and cannot be used in the analysis, which means the common part does not have any effect on the result of the last query.

Better Privacy

In *tracker attacks* an adversary issues different queries involving a particular record to extract its information [3]. Studying records with high privacy balance empowers us to track suspicious queries, endangered records and those who run these queries.

Parallel Execution

Record Based Privacy budgeting approach is also applicable in distributed environments. In distributed environments different nodes can run one computation on their databases in parallel and do their own budget calculation. The nodes only need to communicate to exchange the results.

8.2 Disadvantages

In this section we discuss possible disadvantages of using record based differential privacy instead of the classic model.

Storage

For every record r_i, a data field PB_i is required to store privacy budget. Privacy budget value is decreased for those records that are involved in an aggregation until it reaches zero. It is possible to use another different field to store balance and increase the value until $balance > budget$. In this setting the system is allowed to use the record until $balance < budget$.

Time

In PINQuin, the time needed to answer each query is greater than the time needed for query execution PINQ. For each transformation operation, it is needed to track the list of involved records and for aggregation operations the privacy budget for all involved record needs to be updated.

9 Experimental Results

In this section we present our experimental results concerning the application of PINQuin to compare record based privacy budgeting with the classic method. The case studies have been taken from standard sample projects in PINQ package. We successfully converted `MachineLearning, SocialNetworking` and `TestHarness` to their PINQuin equivalents, but `Visualization` project needed a DryadLINQ data source (search log) that was not available for public usage. Finally we implemented the dynamic queue algorithm that cannot be efficiently implemented using the classic model. Dynamic queue is a kind of database that is frequently updated with fresh records. In what follows we only present our experiments in the `MachineLearning` case study, and we discuss the problem with implementing the dynamic queue in the classic model and demonstrate our solution.

	PINQ Execution time	**PINQuin** Execution time
K-mean	11444759	37498023
Social Network	10703129	131355210
Test Harness	4686407	4953974

Table 5: Comparison between PINQ and PINQuin

Machine Learning, K-means Clustering

To compare different aspects of our framework we decided to do performance analysis on `MachineLearning` project that implements the *K-mean* clustering algorithm. We chose this algorithm because the problem description is well-known and it also uses parallel compositions.

The simple K-mean algorithm used in this project initiated by k random centers, t observations in n dimensional Cartesian coordinate system and returns clusters with updated centers. In Figure 15 cluster centers are marked with bold circles. Each center represents a cluster and in each iteration, observations are assigned to one of the clusters based on distance from centers. Next, each center is moved to the mean of each cluster and a new iteration starts. The algorithm finishes when centers do not change their place.

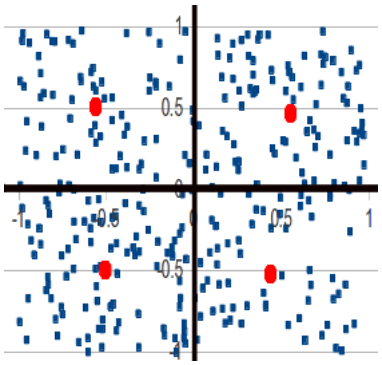

Figure 15: K-mean clustering

As we are dealing with noisy operation we assume the algorithm converges after i iterations. In our experiment, random observations (x,y) are chosen from the following range: $|x| < 1, |y| < 1$. Random centers usually converges to the following four points: (0.5,0.5), (−0.5,0.5), (0.5, − 0.5), (−0.5, − 0.5).

We implemented four versions of the K-mean algorithm with the same setting of parameters: t=10000; n=2; k=4; i=5. The first version of the K-mean algorithm uses `Where()` and `Average()` operations from LINQ. As no noise is added to the calculation the result is used as a reference to compare the accuracy of other algorithms. The second version of the K-mean algorithm uses the `Partition()` and `NoisyAverage()` operations from PINQ. As we partition the observations into clusters and run the queries in parallel, we can increase the accuracy parameter and have more accurate results. The third version uses the `Where()` and `NoisyAverage()` operations from PINQ. As the queries are executed sequentially, we need to decrease the accuracy parameter resulting in less accurate results. Finally the last version of the K-mean algorithm uses the `Where()` and `NoisyAverage()` operations from PINQuin. The result of our experiments shows PINQuin has the same accuracy as when parallel composition is used with complex `Partition()` operation. In Listing 7 a sample execution output for all four algorithms can be seen.

Listing 7 Result of running different implementations

```
-----Centers----
-0.9801 0.5347
0.6397 -0.2794
-0.3733 0.7618
-0.0803 -0.0164
-----LINQ Where----
-0.5763 0.4651
0.5558 -0.4385
0.4206 0.5532
-0.4448 -0.5287
Elapsed Ticks:5020511883
-----PINQ Partition----
-0.5864 0.4634
0.5611 -0.4401
0.4178 0.5676
-0.4381 -0.5254
Elapsed Ticks:827854848
-----PINQ Where----
-0.6234 0.3991
0.5509 -0.4212
0.3952 0.5720
-0.3884 -0.6018
Elapsed Ticks:10576789044
-----PINQuin Where----
-0.6081 0.4692
0.5436 -0.4283
0.4242 0.5659
-0.4393 -0.5002
Elapsed Ticks:2788920243
```

As it can be seen in Table 6, the average amount of noise in the algorithm that uses parallel composition using `Partition()` (in the PINQ columns) is almost the same as the one that uses sequential composition using `Where` in PINQuin (last column), however normal sequential composition using PINQ results in five times as much noise as its parallel implementation.

We used the system timer to measure the execution time of all algorithms. Parallel composition has the lowest execution time[12] as the aggregation operations are executed on subsets of records rather than all the records. While we expected longer execution time when using PINQuin than sequential implementation using PINQ, surprisingly PINQuin had a shorter execution time. We do not know the real reason for this improvement.

[12] A tick is the smallest measurable time unit.

To sum up, with the same privacy budget and accuracy the record based privacy budgeting can answer more queries with the price of longer execution time.

	LINQ	PINQ (Parallel)	PINQ (Sequential)	PINQuin
Execution Time	50205118	8278548	105767890	27889202
Amount of Noise	0	0.019389	0.098496	0.024356

Table 6: Comparison between different implementations of K-mean

Dynamic Queue

As mentioned before, the accuracy of the result decreases when records run out of privacy budget. Record based privacy budgeting is only efficient if queries do not inherently consume all records budget or the number of records that run out of budget is small.

For instance, for a queue (Figure 16) with flow rate R, queue size L and record budget B, it is possible to execute ϵ-differentially private queries with the following rate: $\frac{B \times R}{L \times \epsilon}$. In this scenario, the buffer is constantly refreshed with new records, so possible run out of budget records are replaced with fresh ones. There is no limit on the query execution rate but when it passes the $\frac{B \times R}{L \times \epsilon}$, some records may not participate in the analysis which makes the results inaccurate. This is an important trade off that analysts should think about.

We simulated the dynamic queue with a data structure of `PINQuin<double>`. Two threads, `Analyzer` and `Updater` are used. The `Updater` thread removes old elements from the data set and insert fresh ones into the protected data set. The `Analyzer` can run queries with steady rate of $\frac{B \times R}{L \times \epsilon}$. If the analyst needs more accurate results he has to decrease the query execution rate.

This kind of analysis on network buffers can be used to have a real-time information about network traffic status without violating privacy of any individual person.[13] As another example we can use this method if we are interested to have a real time information about patients that use service from a hospital. Individual's privacy budget avoids privacy breach of individuals who stay longer in the hospital and has more influences on the result of analysis as they participate in higher number of analyses.

When dealing with dynamic databases, current frameworks can only answer the first few queries because these frameworks only see the database as whole, but PINQuin sees records and their privacy impact individually. As a result the queries can be answered as long as the database is updated with fresh records and the accuracy of results depends on the database refresh rate.

[13] Packets should be independent of each other or there should be a way to keep and update privacy usage for each packet's owner.

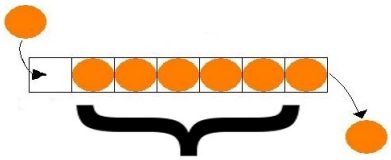

Figure 16: Queue that is updated frequently

Other Applications

PINQuin is also useful for analysts who have access to the database and want to find the best set of queries to run and determine the records that may be the target of privacy breach. This can easily be accomplished by checking those records that consumed more privacy budget.

10 Conclusion

In this master thesis we present *record based differential privacy budgeting* which makes it possible to have more flexible queries. Flexible queries result in higher database utilization and simpler interface to extract statistical information. Using this approach more queries on dynamic databases where data can be added, removed and updated, becomes possible.

Though we have not provided a formal proof, our experiments show that our new method provides the same accuracy and privacy guarantee as the classic method in cost of more storage and time for query execution.

Lastly we introduced PINQuin to demonstrate our idea in action. PINQuin is a framework based on PINQ that uses this new model of privacy budgeting.

Future Work PINQuin did not try to fix the different side channels that are introduced in other research [3]. While we suggest to limit the access to database to trusted analysts, we think there is place for improvement in protections against timing and state side channels. Fuzz uses micro-queries to return a default value in case the calculation takes longer than expected or delay the completion of micro-query in case it finishes earlier than expected. One possible future direction is to use the same mechanism in aggregation operations to protect the underlying data from timing side channels. We are also interested to see if we can block data leakage from state side channels using *.NET appDomains* as suggested in [5].

We are currently working to prove that a record with record privacy budget equals to ϵ in a system that uses record based privacy budgeting has the same level of privacy as the systems that use the classic model of budgeting with total database privacy budget equals to ϵ.

Acknowledgements

This project would not have been possible without support, encouragement and guidance of many individuals. It is with immense gratitude that I acknowledge the support and help of my supervisor, professor Gerardo Schneider from the Department of Computer Science and Engineering at the University of Gothenburg for his encouragement and guidance throughout my thesis. Special thanks also to professor David Sands from the Department of Computer Science and Engineering at University of Chalmers for his time and advices during my thesis and Dr. Frank McSherry and Dr. Arjun Ravi Narayan for taking the time to answer my questions. I also wish to express my gratitude to my beloved family; for their support through the duration of my studies.

Hamid Ebadi Tavallaei, Gothenburg - Sweden, 23 Jan 2013

Bibliography

[1] C. Dwork, A firm foundation for private data analysis, Commun. ACM 54 (1) (2011) 86–95.

[2] F. D. McSherry, Privacy integrated queries: an extensible platform for privacy-preserving data analysis, in: Proceedings of the 2009 ACM SIGMOD International Conference on Management of data, SIGMOD '09, 2009, pp. 19–30.

[3] A. Haeberlen, B. C. Pierce, A. Narayan, Differential privacy under fire, in: Proceedings of the 20th USENIX conference on Security, SEC'11, 2011, pp. 33–33.

[4] I. Roy, S. T. V. Setty, A. Kilzer, V. Shmatikov, E. Witchel, Airavat: security and privacy for mapreduce, in: Proceedings of the 7th USENIX conference on Networked systems design and implementation, NSDI'10, 2010, pp. 20–20.

[5] P. Mohan, A. Thakurta, E. Shi, D. Song, D. Culler, Gupt: privacy preserving data analysis made easy, in: Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data, SIGMOD '12, 2012, pp. 349–360.

[6] B. C. Fung, K. Wang, A. W.-C. Fu, P. S. Yu, Introduction to Privacy-Preserving Data Publishing: Concepts and Techniques, 1st Edition, Chapman & Hall/CRC, 2010.

[7] L. Sweeney, k-anonymity: a model for protecting privacy, Int. J. Uncertain. Fuzziness Knowl.-Based Syst. 10 (5) (2002) 557–570.

[8] Wikipedia, Aol search data leak — wikipedia, the free encyclopedia, [Online; accessed 10-January-2013] (2012).
URL `http://en.wikipedia.org/w/index.php?title=AOL_search_d%ata_leak&oldid=527286678`

[9] A. Narayanan, V. Shmatikov, How to break anonymity of the netflix prize dataset, CoRR abs/cs/0610105.

[10] A. Narayanan, V. Shmatikov, Robust de-anonymization of large sparse datasets, in: Proceedings of the 2008 IEEE Symposium on Security and Privacy, SP '08, 2008, pp. 111–125.

[11] T. Dalenius, Towards a methodology for statistical disclosure control, Statistik Tidskrift 15 (429-444).

[12] C. Dwork, Differential privacy: a survey of results, in: Proceedings of the 5th international conference on Theory and applications of models of computation, TAMC'08, 2008, pp. 1–19.

[13] R. Sarathy, K. Muralidhar, Evaluating laplace noise addition to satisfy differential privacy for numeric data, Trans. Data Privacy 4 (1) (2011) 1–17.

[14] A. Machanavajjhala, D. Kifer, J. Abowd, J. Gehrke, L. Vilhuber, Privacy: Theory meets practice on the map, in: Proceedings of the 2008 IEEE 24th International Conference on Data Engineering, ICDE '08, 2008, pp. 277–286.

[15] Wikipedia, Laplace distribution — wikipedia, the free encyclopedia, [Online; accessed 30-January-2013] (2012).
URL `http://en.wikipedia.org/w/index.php?title=Laplace_distribution&oldid=526109214`

Appendix

Notice: PINQuin is a framework based on PINQ. It tries to be faithful to PINQ, so programs that are written for PINQ can be converted and still be used.

As PINQuin uses APIs, APIs description and code from PINQ, please read the license agreement for PINQ first. Both PINQ and Microsoft research license agreement can be found in the following link : http://research.microsoft.com/en-us/projects/pinq/

Contents

1 Namespace Index

2 Packages

Here are the packages with brief descriptions (if available):

3 Class Index

4 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

5 File Index

6 File List

Here is a list of all files with brief descriptions:

7 Namespace Documentation

8 Package PINQuin

Classes

- class **RecordKeylist**

  This class stores one record and list of its related keys.
- class **RecordKeylistCompare**

  This EqualityComparer only compares two values and doesn't consider the keys.

- struct **BudgetBalance**

  Stores budget and balance for one record.
- class **PINQuin**

9 Class Documentation

10 PINQuin.BudgetBalance< BTYPE > Struct Template Reference

Stores budget and balance for one record.

Public Attributes

- BTYPE **balance**
- BTYPE **budget**

10.1 Detailed Description

template<BTYPE>struct PINQuin::BudgetBalance< BTYPE >

Stores budget and balance for one record.

10.2 Member Data Documentation

10.2.1 template<BTYPE > BTYPE PINQuin.BudgetBalance< BTYPE >.balance

10.2.2 template<BTYPE > BTYPE PINQuin.BudgetBalance< BTYPE >.budget

The documentation for this struct was generated from the following file:

- **PINQuin.cs**

11 PINQuin.PINQuin< T > Class Template Reference

Public Member Functions

- **PINQuin** ()
- **PINQuin** (IQueryable< T > source, double budget)

  Initializes a new instance of the PINQuin.PINQuin'1 class. keys are assigned randomly.
- **PINQuin** (IQueryable< **RecordKeylist**< int, T >> kr, Dictionary< int, **BudgetBalance**< double > > kbb)

Initializes a new instance of the PINQuin.PINQuin'1 class.

- virtual **PINQuin**< S > **NewPINQuin**< **S** > (IQueryable< **RecordKeylist**< int, S >> kr, Dictionary< int, **BudgetBalance**< double > > kbb)

  *Creates a new **PINQuin** (p. 2).*
- virtual **PINQuin**< T > **AsPINQuin** ()
- virtual **PINQuin**< T > **Materialize** ()

  Materializes the contents of a PINQueryable. In principle, this result in no information disclosure, but can be a useful performance optimization for LINQ providers that are not good about spotting common subexpressions.
- **PINQuin**< T > **Insert** (IQueryable< T > dataSource, double budget)

  Inserts a list of records into data set.
- int **Insert** (T item, double budget)

  Inserts one record to data set.
- bool **InsertWithListOfKeys** (T item, List< int > keys)

  Inserts one record with a list of related keys.
- bool **InsertWithKey** (int tmpkey, T item, double budget)

  Inserts a record with user defined key and budget.
- bool **Update** (Expression< Func< T, bool >> predicate, T record)

  Updates records specified by predicate.
- bool **UpdateByKey** (int tmpkey, T record)

  Updates one record that is specified by key.
- bool **Remove** (Expression< Func< T, bool >> predicate)

  Removes records specified by predicate.
- bool **RemoveByKey** (int tmpkey)

  Removes the record that is specified by key.
- void **showBB** ()

  Shows budget, balance pair of values for all records, FOR DEBUGGING PURPOSE ONLY.
- void **Print** ()

  Print information about data stored in instance, FOR DEBUGGING PURPOSE ONLY.
- **PINQuin**< T > **Where** (Expression< Func< T, bool >> predicate)

  Where transformation.
- **PINQuin**< S > **Select**< **S** > (Expression< Func< T, S >> selector)

  Select transformation.
- **PINQuin**< S > **SelectMany**< **S** > (int k, Expression< Func< T, IEnumerable< S >>> selector)

  SelectMany transformation. Each record may only produce a limited number of records.
- **PINQuin**< R > **Join**< **S, K, R** > (IQueryable< S > other, Expression< Func< T, K >> keySelector1, Expression< Func< S, K >> keySelector2, Expression< Func< IGrouping< K, T >, IGrouping< K, S >, R >> resultSelector)

  Join with an unprotected IQueryable.
- **PINQuin**< R > **Join**< **S, K, R** > (**PINQuin**< S > other, Expression< Func< T, K >> keySelector1, Expression< Func< S, K >> keySelector2, Expression< Func< IGrouping< K, T >, IGrouping< K, S >, R >> resultSelector)

*Entrypoint for Join with a **PINQuin*** (p. 2)*. Passes control to the other **PINQuin*** (p. 2) *to expose its IQueryable.*

- virtual **PINQuin**< R > **JoinHelper**< **S, K, R** > (List< **RecordKeylist**< int, S > > otherKeyRecord, Expression< Func< T, K >> keySelector1, Expression< Func< S, K >> keySelector2, Expression< Func< IGrouping< K, S >, IGrouping< K, T >, R >> resultSelector)

  *Helper method for Join. Invokes first **PINQuin*** (p. 2)*'s join on its unprotected data.*

- **PINQuin**< R > **Join**< **S, K, R** > (IQueryable< S > other, Expression< Func< T, K >> keySelector1, Expression< Func< S, K >> keySelector2, int bound1, int bound2, Expression< Func< T, S, R >> resultSelector)

  Bounded Join with unprotected IQueryable. The join imposes a limit on the number of records from each data set mapping to each key.

- **PINQuin**< R > **Join**< **S, K, R** > (**PINQuin**< S > other, Expression< Func< T, K >> keySelector1, Expression< Func< S, K >> keySelector2, int bound1, int bound2, Expression< Func< T, S, R >> resultSelector)

  *Entrypoint for Join with a **PINQuin*** (p. 2)*. Passes control to the other **PINQuin*** (p. 2) *to expose its list of KeyRecord.*

- virtual **PINQuin**< R > **JoinHelper**< **S, K, R** > (List< **RecordKeylist**< int, S > > otherKeyRecord, Expression< Func< T, K >> keySelector1, Expression< Func< S, K >> keySelector2, int bound1, int bound2, Expression< Func< S, T, R >> resultSelector)

  *Bounded Join with unprotected list of **RecordKeylist*** (p. 27) *intended to protect it. The join imposes a limit on the number of records from each data set mapping to each key.*

- **PINQuin**< R > **GroupJoin**< **S, K, R** > (IQueryable< S > other, Expression< Func< T, K >> keySelector1, Expression< Func< S, K >> keySelector2, Expression< Func< T, IEnumerable< S >, R >> resultSelector)

  GroupJoin with an unprotected IQueryable.

- **PINQuin**< R > **GroupJoinx**< **S, K, R** > (IQueryable< S > other, Expression< Func< T, K >> keySelector1, Expression< Func< S, K >> keySelector2, Expression< Func< IGrouping< K, T >, IEnumerable< S >, R >> resultSelector)

  LINQ GroupJoin with an unprotected IQueryable, and result selector that expects pairs of groups.

- **PINQuin**< R > **GroupJoin**< **S, K, R** > (**PINQuin**< S > other, Expression< Func< T, K >> keySelector1, Expression< Func< S, K >> keySelector2, Expression< Func< IGrouping< K, T >, IEnumerable< S >, R >> resultSelector)

  *GroupJoin entry point with protected **PINQuin*** (p. 2)*.*

- virtual **PINQuin**< R > **GroupJoinHelper**< **S, K, R** > (List< **RecordKeylist**< int, S > > otherKeyRecord, Expression< Func< T, K >> keySelector1, Expression< Func< S, K >> keySelector2, Expression< Func< IGrouping< K, S >, IEnumerable< T >, R >> resultSelector)

  *GroupJoin helper method. Passes control to the other **PINQuin*** (p. 2)*, with an unprotected data set.*

- **PINQuin**< IGrouping< K, T > > **GroupBy**< **K** > (Expression< Func< T, K >> keySelector)

  GroupBy operation.

- **PINQuin**< T > **Distinct** ()

  Restuns distinct records.
- **PINQuin**< T > **Distinct** (int k)

  Distinct up to k elements.
- **PINQuin**< T > **Distinct**< **K** > (int k, Expression< Func< T, K >> keySelector)

  Distinct with key selector.
- **PINQuin**< T > **Union** (IQueryable< T > other)

  Union with an unprotected IQueryable.
- **PINQuin**< T > **Union** (**PINQuin**< T > other)

  Union entry point.
- virtual **PINQuin**< T > **UnionHelper** (List< **RecordKeylist**< int, T > > otherKeyRecord)

  *Union Helper. Passes control to other **PINQuin** (p. 2) with an unprotected IQueryable.*
- **PINQuin**< T > **Intersect** (IQueryable< T > other)

  LINQ Intersect with an unprotected IQueryable.
- **PINQuin**< T > **Intersect** (**PINQuin**< T > other)

  LINQ Intersect entry point.
- virtual **PINQuin**< T > **IntersectHelper** (List< **RecordKeylist**< int, T > > otherKeyRecord)

  *Intersect helper function. Passes control to the other **PINQuin** (p. 2) with an unprotected IQueryable.*
- **PINQuin**< T > **Except** (IQueryable< T > other)

  LINQ Except with an unprotected IQueryable.
- **PINQuin**< T > **Except** (**PINQuin**< T > other)

  *PINQ Except entry point. Passes control to the other **PINQuin** (p. 2).*
- virtual **PINQuin**< T > **ExceptHelper** (List< **RecordKeylist**< int, T > > otherKeyRecord)

  *Except helper. Passes control to other **PINQuin** (p. 2) with an unprotected IQueryable.*
- **PINQuin**< T > **Concat** (IQueryable< T > other)

  Concat with an unprotected IQueryable.
- **PINQuin**< T > **Concat** (**PINQuin**< T > other)

  Concat entry point.
- virtual **PINQuin**< T > **ConcatHelper** (List< **RecordKeylist**< int, T > > otherKeyRecord)

  *Concat helper method. Passes control to the other **PINQuin** (p. 2) with an unprotected RecordKeyList.*
- virtual Dictionary< K, **PINQuin** < T > > **Partition**< **K** > (K[] keys, Expression< Func< T, K >> keyFunc)

  *Partitions the **PINQuin** (p. 2) into a set of PINQuins, one for each of the provided keys.*
- **PINQuin**< T > **Take** (int count)

  Take transformation.
- **PINQuin**< T > **Skip** (int count)

  Skip transformation.

- double **NoisyCount** (double epsilon)

  Counts the number of tuples in the source data set, with noise added for privacy.
- double **NoisySum** (double epsilon, Expression< Func< T, double >> function)

  Computes a noisy sum resulting from the application of function to each record. The function is first tested again the expression visitor, and the output of each invocation is clamped to [-1,+1].
- double **NoisyAverage** (double epsilon, Expression< Func< T, double >> function)

  Computes a noisy average resulting from the application of function to each record. The function is tested against the expression visitor, and after application each output value is clamped to [-1,+1].
- double **NoisyAggregate** (double epsilon, Expression< Func< T, double >> function, double seed, Expression< Func< double, double, double >> aggregationFuction, double sensivity)

  General purpose noisy aggregation function.
- double **NoisyOrderStatistic** (double epsilon, double fraction, Expression< Func< T, double >> function)

  Computes a value in [0,1] that splits the input at roughly the intended fraction.
- double **NoisyMedian** (double epsilon, Expression< Func< T, double >> function)

  Computes a value in [0,1] that splits the data approximately in half. Uses NoisyOrderStatistic.
- R **ExponentialMechanism**< **R** > (double epsilon, IQueryable< R > range, Expression< Func< T, R, double >> scoreFunc)

  Selects an element of range using the provided score function to evaluate each option against the tuples in the data set. Output is selected with probability proportional to the exponential of epsilon times the summed score function applied to each tuple in the data set.

Protected Attributes

- Func< Expression, Expression > **rewrite**

Static Protected Attributes

- static System.Random **random** = new System.Random()

  Random number generator for all randomized query responses. Consider strengthening as appopriate.

template<T> class PINQuin::PINQuin< T >

11.1 Member Function Documentation

11.1.1 template<T > virtual PINQuin<T> PINQuin.PINQuin< T >.AsPINQuin ( ) [virtual]

11.1.2 template<T > PINQuin<T> PINQuin.PINQuin< T >.Concat (IQueryable< T > *other*)

Concat with an unprotected IQueryable.

Parameters

other	Other IQueryable

Returns

PINQuin (p. 2) containing the concatenation of the two data sets.

11.1.3 template<T > PINQuin<T> PINQuin.PINQuin< T >.Concat (PINQuin< T > *other*)

Concat entry point.

Parameters

other	Other **PINQuin** (p. 2)

Returns

Passes control to other **PINQuin** (p. 2). Intends to return a **PINQuin** (p. 2) containing their concatenation.

11.1.4 template<T > virtual PINQuin<T> PINQuin.PINQuin< T >.ConcatHelper (List< RecordKeylist< int, T > > *otherKeyRecord*) [virtual]

Concat helper method. Passes control to the other **PINQuin** (p. 2) with an unprotected RecordKeyList.

Parameters

first	Other **PINQuin** (p. 2)

Returns

Passes control to other **PINQuin** (p. 2). Intends to return a **PINQuin** (p. 2) containing their concatenation.

11.1.5 template<T > PINQuin<T> PINQuin.PINQuin< T >.Distinct ( )

Restuns distinct records.

Returns

PINQuin (p. 2) containing the distinct set of elements

11.1.6 template<T > PINQuin<T> PINQuin.PINQuin< T >.Distinct (int *k*)

Distinct up to k elements.

Parameters

k	max number of elements

Returns

Set of at most k of each elements

11.1.7 template<T > PINQuin<T> PINQuin.PINQuin< T >.Distinct< K > (int *k*, Expression< Func< T, K >> *keySelector*)

Distinct with key selector.

Template Parameters

K	Key type

Parameters

k	max number of elements
keySelector	key selector to distinct by

Returns

Distinct with key function used rather than the elements

11.1.8 template<T > PINQuin<T> PINQuin.PINQuin< T >.Except (IQueryable< T > *other*)

LINQ Except with an unprotected IQueryable.

Parameters

other	Other IQueryable

Returns

PINQuin (p. 2) containing all records except those in other.

11.1.9 template<T > PINQuin<T> PINQuin.PINQuin< T >.Except (PINQuin< T > *other*)

PINQ Except entry point. Passes control to the other **PINQuin** (p. 2).

Parameters

other	Other **PINQuin** (p. 2)

Returns

Passes control to the other **PINQuin** (p. 2). Intends to return a **PINQuin** (p. 2) containing all records except those in other.

11.1.10 template<T > virtual PINQuin<T> PINQuin.PINQuin< T >.ExceptHelper (List< RecordKeylist< int, T > > *otherKeyRecord*) [virtual]

Except helper. Passes control to other **PINQuin** (p. 2) with an unprotected IQueryable.

Parameters

first	Other **PINQuin** (p. 2)

Returns

Passes control to the other **PINQuin** (p. 2). Intends to return a **PINQuin** (p. 2) containing all records in other, except records in this.

11.1.11 template<T > R PINQuin.PINQuin< T >.ExponentialMechanism< R > (double *epsilon,* IQueryable< R > *range,* Expression< Func< T, R, double >> *scoreFunc*)

Selects an element of range using the provided score function to evaluate each option against the tuples in the data set. Output is selected with probability proportional to the exponential of epsilon times the summed score function applied to each tuple in the data set.

Template Parameters

R	Type of the output.

Parameters

epsilon	Describes the accuracy of the mechanism, by dampening the influence of the score function, as well as the privacy loss.
range	Set of possible outputs.
scoreFunc	Function that scores each of the tuples of the data set against each of the possible results.

Returns

Returns a random element of range, with probability exponentially favoring those elements that score well in aggregate with tuples in the data set.

11.1.12 template<T > PINQuin<IGrouping<K, T> > PINQuin.PINQuin< T >.GroupBy< K > (Expression< Func< T, K >> *keySelector*)

GroupBy operation.

Template Parameters

K	Key type

Parameters

keySelector	Key selector

Returns

PINQuin (p. 2) containing a list of groups, one for each observed key, of the records mapping to that key.

11.1.13 template<T > PINQuin<R> PINQuin.PINQuin< T >.GroupJoin< S, K, R > (IQueryable< S > *other,* Expression< Func< T, K >> *keySelector1,* Expression< Func< S, K >> *keySelector2,* Expression< Func< T, IEnumerable< S >, R >> *resultSelector*)

GroupJoin with an unprotected IQueryable.

Template Parameters

S	Other data type
K	Key type
R	Result type

Parameters

other	Other IQueryable
keySelector1	This key selector
keySelector2	Other key selector
result-Selector	Result selector

Returns

PINQuin (p. 2) containing the GroupJoin with the unprotected IQueryable.

11.1.14 template<T > PINQuin<R> PINQuin.PINQuin< T >.GroupJoin< S, K, R > (PINQuin< S > *other,* Expression< Func< T, K >> *keySelector1,* Expression< Func< S, K >> *keySelector2,* Expression< Func< IGrouping< K, T >, IEnumerable< S >, R >> *resultSelector*)

GroupJoin entry point with protected **PINQuin** (p. 2).

Template Parameters

S	Other data type
K	Key type
R	Result type

Parameters

other	Other data set
keySelector1	This key selector
keySelector2	Other key selector
result-Selector	Result selector

Returns

Passes control to other **PINQuin** (p. 2). Intends to result in the GroupJoin of the two, each GroupBy'd their key selectors.

11.1.15 template<T > virtual PINQuin<R> PINQuin.PINQuin< T >.GroupJoinHelper< S, K, R > (List< RecordKeylist< int, S > > *otherKeyRecord,* Expression< Func< T, K >> *keySelector1,* Expression< Func< S, K >> *keySelector2,* Expression< Func< IGrouping< K, S >, IEnumerable< T >, R >> *resultSelector*) [virtual]

GroupJoin helper method. Passes control to the other **PINQuin** (p. 2), with an unprotected data set.

Template Parameters

S	Other data type
K	Key type
R	Result type

Parameters

other	Other data set
keySelector1	This key selector
keySelector2	Other key selector
result-Selector	Result selector

Returns

Passes control to the other **PINQuin** (p. 2). Intends to return the GroupJoin between the two, each GroupBy'd their key selectors.

11.1.16 template< T > PINQuin< R > PINQuin.PINQuin< T >.GroupJoinx< S, K, R > (IQueryable< S > *other,* Expression< Func< T, K >> *keySelector1,* Expression< Func< S, K >> *keySelector2,* Expression< Func< IGrouping< K, T >, IEnumerable< S >, R >> *resultSelector*)

LINQ GroupJoin with an unprotected IQueryable, and result selector that expects pairs of groups.

Template Parameters

S	Other data type
K	Key type
R	Result type

Parameters

other	Other data set
keySelector1	This key selector
keySelector2	Other key selector
result-Selector	Result selector

Returns

PINQuin (p. 2) containing the GroupJoin with the unprotected IQueryable, the first having been GroupBy'd its key selector.

11.1.17 template<T > PINQuin<T> PINQuin.PINQuin< T >.Insert (IQueryable< T > *dataSource,* double *budget*)

Inserts a list of records into data set.

Parameters

other	list of records.
budget	The amount of privacy budget that will be assigned to each added record

11.1.18 template<T > int PINQuin.PINQuin< T >.Insert (T *item,* double *budget*)

Inserts one record to data set.

Parameters

item	the record
budget	privacy budget for the record

11.1.19 template<T > bool PINQuin.PINQuin< T >.InsertWithKey (int *tmpkey,* T *item,* double *budget*)

Inserts a record with user defined key and budget.

Returns

Returns True if successful.

Parameters

tmpkey	assigned key for the record
item	the record
budget	privacy budget for the record

11.1.20 template<T > bool PINQuin.PINQuin< T >.InsertWithListOfKeys (T *item,* List< int > *keys*)

Inserts one record with a list of related keys.

Returns

Returns True if successful

Parameters

item	the record
keys	list of related keys

Exceptions

System.Argument-OutOfRange-Exception	Exception if insertion is not possible

11.1.21 template<T > PINQuin<T> PINQuin.PINQuin< T >.Intersect (IQueryable< T > *other*)

LINQ Intersect with an unprotected IQueryable.

Parameters

other	Other IQueryable

Returns

PINQuin (p. 2) containing the intersection of the two data sets.

11.1.22 template<T > PINQuin<T> PINQuin.PINQuin< T >.Intersect (PINQuin< T > *other*)

LINQ Intersect entry point.

Parameters

other	Otehr **PINQuin** (p. 2)

Returns

Passes control to the other **PINQuin** (p. 2). Intends to return a **PINQuin** (p. 2) containing their intersection.

11.1.23 template<T > virtual PINQuin<T> PINQuin.PINQuin< T >.IntersectHelper (List< RecordKeylist< int, T > > *otherKeyRecord*) [virtual]

Intersect helper function. Passes control to the other **PINQuin** (p. 2) with an unprotected IQueryable.

Parameters

first	Other **PINQuin** (p. 2)

Returns

Passes control to the other **PINQuin** (p. 2). Intends to return a **PINQuin** (p. 2) containing their intersection.

11.1.24 template< T > PINQuin< R > PINQuin.PINQuin< T >.Join< S, K, R > (IQueryable< S > *other,* Expression< Func< T, K >> *keySelector1,* Expression< Func< S, K >> *keySelector2,* Expression< Func< IGrouping< K, T >, IGrouping< K, S >, R >> *resultSelector*)

Join with an unprotected IQueryable.

Template Parameters

S	Other record type
K	Key type
R	Result type

Parameters

other	Second data set
keySelector1	First key selector
keySelector2	Second key selector
result-Selector	Result selector

Returns

PINQuin (p. 2) containing the Join of the two data sets, each first GroupBy'd using their key selector functions.

11.1.25 template< T > PINQuin< R > PINQuin.PINQuin< T >.Join< S, K, R > (PINQuin< S > *other,* Expression< Func< T, K >> *keySelector1,* Expression< Func< S, K >> *keySelector2,* Expression< Func< IGrouping< K, T >, IGrouping< K, S >, R >> *resultSelector*)

Entrypoint for Join with a **PINQuin** (p. 2). Passes control to the other **PINQuin** (p. 2) to expose its IQueryable.

Template Parameters

S	Second data type
K	Key type
R	Result type

Parameters

other	Second **PINQuin** (p. 2)
keySelector1	First key selector
keySelector2	Second key selector
result-Selector	Result selector

Returns

Passes control to other **PINQuin** (p. 2). Intends to return a **PINQuin** (p. 2) containing the Join of the two data sets, after each is GroupBy'd using their key selectors.

11.1.26 template< T > PINQuin< R > PINQuin.PINQuin< T >.Join< S, K, R > (IQueryable< S > *other,* Expression< Func< T, K >> *keySelector1,* Expression< Func< S, K >> *keySelector2,* int *bound1,* int *bound2,* Expression< Func< T, S, R >> *resultSelector*)

Bounded Join with unprotected IQueryable. The join imposes a limit on the number of records from each data set mapping to each key.

Template Parameters

S	Other data type
K	Key type
R	Result type

Parameters

other	Other data set
keySelector1	First key selector
keySelector2	Secord key selector
bound1	bound on per-key records in first data set
bound2	bound on pre-key records in second data set
result-Selector	result selector

Returns

PINQuin (p. 2) containing the LINQ JOIN of the first bound1 and bound2 records for each key, from the two data sets.

11.1.27 template<T > PINQuin<R> PINQuin.PINQuin< T >.Join< S, K, R > (PINQuin< S > *other,* Expression< Func< T, K >> *keySelector1,* Expression< Func< S, K >> *keySelector2,* int *bound1,* int *bound2,* Expression< Func< T, S, R >> *resultSelector*)

Entrypoint for Join with a **PINQuin** (p. 2). Passes control to the other **PINQuin** (p. 2) to expose its list of KeyRecord.

Template Parameters

S	Second data type
K	Key type
R	Result type

Parameters

other	Second **PINQuin** (p. 2)
keySelector1	First key selector
keySelector2	Second key selector
bound1	bound on per-key records in first data set
bound2	bound on pre-key records in second data set
result-Selector	Result selector

Returns

Passes control to other **PINQuin** (p. 2). Intends to return a **PINQuin** (p. 2) containing the Join of the two data sets, after each is GroupBy'd using their key selectors.

11.1.28 template<T > virtual PINQuin<R> PINQuin.PINQuin< T >.JoinHelper< S, K, R > (List< RecordKeylist< int, S > > *otherKeyRecord,* Expression< Func< T, K >> *keySelector1,* Expression< Func< S, K >> *keySelector2,* Expression< Func< IGrouping< K, S >, IGrouping< K, T >, R >> *resultSelector*) [virtual]

Helper method for Join. Invokes first **PINQuin** (p. 2)'s join on its unprotected data.

Template Parameters

S	First data type
K	Key type
R	Result type

Parameters

other	First data set
keySelector1	This key selector
keySelector2	Other key selectar
result-Selector	Result selector

Returns

Invokes the other **PINQuin** (p. 2)'s Join method with unprotected data, and returns its result.

11.1.29 template<T > virtual PINQuin<R> PINQuin.PINQuin< T >.JoinHelper< S, K, R > (List< RecordKeylist< int, S > > *otherKeyRecord,* Expression< Func< T, K >> *keySelector1,* Expression< Func< S, K >> *keySelector2,* int *bound1,* int *bound2,* Expression< Func< S, T, R >> *resultSelector*) [virtual]

Bounded Join with unprotected list of **RecordKeylist** (p. 27) intended to protect it. The join imposes a limit on the number of records from each data set mapping to each key.

Template Parameters

S	Other data type
K	Key type
R	Result type

Parameters

otherKey-Record	Other data set
keySelector1	First key selector
keySelector2	Secord key selector
bound1	bound on per-key records in first data set
bound2	bound on pre-key records in second data set
result-Selector	result selector

Returns

PINQuin (p. 2) containing the LINQ JOIN of the first bound1 and bound2 records for each key, from the two data sets.

11.1.30 template<T > virtual PINQuin<T> PINQuin.PINQuin< T >.Materialize ( ) [virtual]

Materializes the contents of a PINQueryable. In principle, this result in no information disclosure, but can be a useful performance optimization for LINQ providers that are not good about spotting common subexpressions.

Returns

11.1.31 template<T > virtual PINQuin<S> PINQuin.PINQuin< T >.NewPINQuin< S > (IQueryable< RecordKeylist< int, S >> *kr,* Dictionary< int, BudgetBalance< double > > *kbb*) [virtual]

Creates a new **PINQuin** (p. 2).

Returns

The new **PINQuin** (p. 2).

Parameters

kr	**RecordKeylist** (p. 27)
kbb	IF WE ARE CREATING A NEW **PINQuin** (p. 2) with a new TYPE, KEY_BUDGET_BALANCE is needed to be passed, otherwise null can be passed NEW TYPES ARE USUALLY INTRODUCED IN SEELCT,SELECYMANY, GROUPBY, AND JOIN OPERATIONS

Template Parameters

S	The type of new **PINQuin** (p. 2).

11.1.32 template<T > double PINQuin.PINQuin< T >.NoisyAggregate (double *epsilon,* Expression< Func< T, double >> *function,* double *seed,* Expression< Func< double, double, double >> *aggregationFuction,* double *sensivity*)

General purpose noisy aggregation function.

Returns

The result of aggregation

Parameters

epsilon	Determines the accuracy of the average, and the privacy lost.
function	function that applies to each record.
seed	starting seed.
aggregation-Fuction	Aggregation fuction.

Exceptions

Exception	Represents errors that occur during application execution.

11.1.33 template<T > double PINQuin.PINQuin< T >.NoisyAverage (double *epsilon,* Expression< Func< T, double >> *function*)

Computes a noisy average resulting from the application of function to each record. The function is tested against the expression visitor, and after application each output value is clamped to [-1,+1].

Parameters

epsilon	Determines the accuracy of the average, and the privacy lost
function	Function to apply to each tuple, yield a number in [0,1]

Returns

The average of the [0,1] values described by function, plus Laplace(2.0/epsilon)/-Count.

11.1.34 template<T > double PINQuin.PINQuin< T >.NoisyCount (double *epsilon*)

Counts the number of tuples in the source data set, with noise added for privacy.

Parameters

epsilon	The accuracy of the associated noise, influencing the privacy lost.

Returns

The count of the source data set, plus Laplace(1.0/epsilon).

11.1.35 template<T > double PINQuin.PINQuin< T >.NoisyMedian (double *epsilon,* Expression< Func< T, double >> *function*)

Computes a value in [0,1] that splits the data approximately in half. Uses NoisyOrderStatistic.

Parameters

epsilon	Amount of privacy lost. Controls the sensitivity of the computation.
function	Function to produce values in [0,1] from the source tuples.

Returns

Randomly selected element of [0,1] with exponential bias towards those that spilt the data into equal sized parts.

11.1.36 template<T > double PINQuin.PINQuin< T >.NoisyOrderStatistic (double *epsilon,* double *fraction,* Expression< Func< T, double >> *function*)

Computes a value in [0,1] that splits the input at roughly the intended fraction.

Parameters

epsilon	Amount of privacy lost. Controls the sensitivity of the computation.
fraction	Value in [0,1] indicating the fraction of data to be split. 0.5 would be median.
function	Function to produce values in [0,1] from the source tuples.

Returns

Randomly selected element of [0,1] with exponential bias towards values that split with the appropriate balance.

11.1.37 template<T > double PINQuin.PINQuin< T >.NoisySum (double *epsilon,* Expression< Func< T, double >> *function*)

Computes a noisy sum resulting from the application of function to each record. The function is first tested again the expression visitor, and the output of each invocation is clamped to [-1,+1].

Parameters

epsilon	Determines the accuracy of the sum, and privacy lost.
function	Function to be applied to each record. Results are clamped to [-1.-0,+1.0].

Returns

The clamped sums of the application of function to each record, plus Laplace(1.-0/epsilon).

11.1.38 template<T > virtual Dictionary<K, PINQuin<T> > PINQuin.PINQuin< T >.Partition< K > (K[] *keys,* Expression< Func< T, K >> *keyFunc*) [virtual]

Partitions the **PINQuin** (p. 2) into a set of PINQuins, one for each of the provided keys.

Template Parameters

K	Type of the keys.

Parameters

keys	Explicit set of possible key values.
keyFunc	Function that yields the key associated with a record.

Returns

An Dictionary mapping each key to a **PINQuin** (p. 2) of records from the source data set that yield the key.

11.1.39 template<T > PINQuin.PINQuin< T >.PINQuin ( )

11.1.40 template<T > PINQuin.PINQuin< T >.PINQuin (IQueryable< T > *source,* double *budget*)

Initializes a new instance of the PINQuin.PINQuin‘1 class. keys are assigned randomly.

Parameters

source	Data source
budget	the amount of budget that will be assigned to all records in data set

11.1.41 template<T > PINQuin.PINQuin< T >.PINQuin (IQueryable< RecordKeylist< int, T >> *kr,* Dictionary< int, BudgetBalance< double > > *kbb*)

Initializes a new instance of the PINQuin.PINQuin‘1 class.

Parameters

kr	a list of records, key list pair
kbb	KeyBudgetBalance list keeps track of privacy accounting

11.1.42 template<T > void PINQuin.PINQuin< T >.Print ( )

Print information about data stored in instance, FOR DEBUGGING PURPOSE ONLY.

11.1.43 template<T > bool PINQuin.PINQuin< T >.Remove (Expression< Func< T, bool >> *predicate*)

Removes records specified by predicate.

Parameters

predicate	Choose which records should be updated.

11.1.44 template<T > bool PINQuin.PINQuin< T >.RemoveByKey (int *tmpkey*)

Removes the record that is specified by key.

Returns

Returns True if update is successful.

Parameters

tmpkey	The key

11.1.45 template<T > PINQuin<S> PINQuin.PINQuin< T >.Select< S > (Expression< Func< T, S >> *selector*)

Select transformation.

Template Parameters

S	Result type of the underlying records.

Parameters

selector	Record-to-record transformation

Returns

PINQuin (p. 2) containing the transformations of records in the source data set.

11.1.46 template<T > PINQuin<S> PINQuin.PINQuin< T >.SelectMany< S > (int *k*, Expression< Func< T, IEnumerable< S >>> *selector*)

SelectMany transformation. Each record may only produce a limited number of records.

Template Parameters

S	Result type of the underlying records.

Parameters

k	Upper bound on number of records produced by each input record.
selector	Record-to-RecordList transformation

Returns

PINQuin (p. 2) containing the first k records produced from each source record.

11.1.47 template<T > void PINQuin.PINQuin< T >.showBB ( )

Shows budget, balance pair of values for all records, FOR DEBUGGING PURPOSE ONLY.

11.1.48 template<T > PINQuin<T> PINQuin.PINQuin< T >.Skip (int *count*)

Skip transformation.

Parameters

count	Number of records to skip

Returns

PINQuin (p. 2) containing all but the first count records.

11.1.49 template<T > PINQuin<T> PINQuin.PINQuin< T >.Take (int *count*)

Take transformation.

Parameters

count	Number of records to return

Returns

PINQuin (p. 2) containing the first count records.

11.1.50 template<T > PINQuin<T> PINQuin.PINQuin< T >.Union (IQueryable< T > *other*)

Union with an unprotected IQueryable.

Parameters

other	Other IQueryable

Returns

PINQuin (p. 2) containing the union of the two data sets.

11.1.51 template<T > PINQuin<T> PINQuin.PINQuin< T >.Union (PINQuin< T > *other*)

Union entry point.

Parameters

other	Other **PINQuin** (p. 2)

Returns

Passes control to other **PINQuin** (p. 2). Intends to return a **PINQuin** (p. 2) containing their union.

11.1.52 template<T > virtual PINQuin<T> PINQuin.PINQuin< T >.UnionHelper (List< RecordKeylist< int, T > > *otherKeyRecord*) [virtual]

Union Helper. Passes control to other **PINQuin** (p. 2) with an unprotected IQueryable.

Parameters

first	Other **PINQuin** (p. 2)

Returns

Passes control to other **PINQuin** (p. 2). Intends to return a **PINQuin** (p. 2) containing their union.

11.1.53 template<T > bool PINQuin.PINQuin< T >.Update (Expression< Func< T, bool >> *predicate,* T *record*)

Updates records specified by predicate.

Parameters

predicate	Choose which records should be updated.
record	the record

11.1.54 template<T > bool PINQuin.PINQuin< T >.UpdateByKey (int *tmpkey,* T *record*)

Updates one record that is specified by key.

Returns

Returns True if update is successful.

Parameters

tmpkey	The key
record	The record

11.1.55 template<T > PINQuin<T> PINQuin.PINQuin< T >.Where (Expression< Func< T, bool >> *predicate*)

Where transformation.

Parameters

predicate	boolean predicate applied to each record

Returns

PINQuin (p. 2) containing the subset of records satisfying the input predicate.

11.2 Member Data Documentation

11.2.1 template<T > System.Random PINQuin.PINQuin< T >.random = new System.Random() [static, protected]

Random number generator for all randomized query responses. Consider strengthening as appopriate.

11.2.2 template<T > Func<Expression, Expression> PINQuin.PINQuin< T >.rewrite [protected]

The documentation for this class was generated from the following file:

- **PINQuin.cs**

12 PINQuin.RecordKeylist< KT, T > Class Template Reference

This class stores one record and list of its related keys.

Properties

- T **record** [get, set]
- List< KT > **key** [get, set]

12.1 Detailed Description

template<KT, T>class PINQuin::RecordKeylist< KT, T >

This class stores one record and list of its related keys.

12.2 Property Documentation

12.2.1 template<KT , T > List<KT> PINQuin.RecordKeylist< KT, T >.key [get, set]

12.2.2 template<KT , T > T PINQuin.RecordKeylist< KT, T >.record [get, set]

The documentation for this class was generated from the following file:

- **PINQuin.cs**

13 PINQuin.RecordKeylistCompare< KT, T > Class Template - Reference

This EqualityComparer only compares two values and doesn't consider the keys.

Public Member Functions

- bool **Equals** (**RecordKeylist**< KT, T > x, **RecordKeylist**< KT, T > y)
- int **GetHashCode** (**RecordKeylist**< KT, T > product)

13.1 Detailed Description

template<KT, T>class PINQuin::RecordKeylistCompare< KT, T >

This EqualityComparer only compares two values and doesn't consider the keys.

13.2 Member Function Documentation

13.2.1 template<KT , T > bool PINQuin.RecordKeylistCompare< KT, T >.Equals (RecordKeylist< KT, T > *x,* RecordKeylist< KT, T > *y*)

13.2.2 template<KT , T > int PINQuin.RecordKeylistCompare< KT, T >.GetHashCode (RecordKeylist< KT, T > *product*)

The documentation for this class was generated from the following file:

- **PINQuin.cs**

14 File Documentation

15 PINQuin.cs File Reference

Classes

- class **PINQuin.RecordKeylist< KT, T >**
  This class stores one record and list of its related keys.
- class **PINQuin.RecordKeylistCompare< KT, T >**
  This EqualityComparer only compares two values and doesn't consider the keys.
- struct **PINQuin.BudgetBalance< BTYPE >**
  Stores budget and balance for one record.
- class **PINQuin.PINQuin< T >**

Packages

- package **PINQuin**

Variables

- using **System**

15.1 Variable Documentation

15.1.1 using System