Privacy System Design Proposal

Michael Shoemate

September 20, 2019

Abstract

A privacy preserving system design is proposed, containing abstractions for a statistical analysis and differentially private release. The statistical analysis is a computational graph and privacy definition. The system has three layers, one each for analysis construction, validation, and execution. Parsers and bindings may be written for analysis construction, and runtimes for analysis execution. Analysis verification is centralized in a core C++ library. The primary runtime is written in C++.

1 Overview

Goals The goal of this system is to provide a flexible framework on which

- Researchers can contribute new algorithms flexible enough to encompass a broad range of topics and approaches to differential privacy,
- conceptually organized enough that peer researchers and/or automated validation systems can more easily review contributed work
- and which allows both Analysts using the library and System Builders creating applications to make use of vetted differentially private algorithms with a low bar with languages native to their workflow.

2 Abstractions

2.1 Analysis

To allow for pre- and post-processing, nested composition and code modularity, an analysis is a computational graph of instances of components. Each component in the analysis graph is either an operator or statistic. An operator may be a transformation, subset or join. The analysis conforms to either a JSON or protobuf schema. No data is stored in the analysis, it is only stored in the release.

2.2 Release

The second primary abstraction is the release. The release is a set of values that are associated with nodes in the accompanying analysis. Entries in a release include the initial variable bounds, the record count and estimated statistics. The types of released statistics may be numeric, string, or even function-valued, depending on the mechanism. Relevant information for a released statistic include the corresponding node id, value, batch, whether the release came from the user or from the runtime, and whether the value is privatized.

In some situations it can be useful to evaluate a computational graph to release private data. Three such examples are for evaluating loss functions, filters, and executing graphs across multiple runtimes.

3 Components

A list of components is available at https://bit.ly/privacy_components. This list is still in-progress.

3.1 Operators

Operators, or manipulations, include transformations, subsets, aggregations and joins. Analyses using manipulations are validated using Lipschitz constants in a "stability" framework. Manipulation primitives may also be reused to define arbitrary objective functions for optimization, for filters, or to describe a differentially private release of a function.

3.2 Mechanisms

Mechanisms are building blocks used by statistics, and if placed in an analysis graph, are not capable of privatizing data on their own. Example mechanisms include the laplace mechanism and exponential mechanism.

3.3 Statistics

A statistic contains several components. For example, a "Mean" may be composed of "Sum" and "Divide" components, and a "Differentially Private Mean" composed of "Impute", "Clip", "Mean" and "Laplace" components. A statistic is the only kind of component that can privatize data.

3.4 Miscellaneous

A "Constant" component propagates a value included from a release. A "Literal" component propagates a value included in its constructor. These components are easily hidden from the user.

4 System Layers

4.1 Analysis Construction

An analysis consists of a computational graph and a privacy definition. The computational graph may be constructed via a graphical interface, manually via language bindings, or automatically via a parser. Any such tools emit an analysis and optionally a partial release. The privacy definition is simply a string label.

4.2 Analysis Validation

The C++ validator has two tasks- checking the graph and computing meta-statistics.

The validator ensures every path through an analysis graph passes through a privatizing statistic, and that the graph is executable (static type checking). Note that mechanisms are not capable of privatizing data alone; released data must pass through curated plans that include clipping, imputation, and have sensible aggregation/mechanism pairings. In addition to these checks, all primitives in an analysis must support the same privacy definition. For example, resampling may not be supported by concentrated privacy. Optionally checks can be made to ensure N is not being manipulated by the user through the computational graph, by enforcing the the child of N parameters to be of type "Constant".

The validator may also compute meta-statistics (accuracy, disclosure risk measures, confidences) from an analysis and partial release, as well as overall epsilon after applying known composition theorems.

4.3 Analysis Execution

The execution layer is the only layer with data access. The execution layer takes an analysis and optionally a partial release, and emits a release. Including the prior release will make sources of randomness across batches deterministic. Protection against timing attacks may be implemented in the execution layer.

An implementation of an execution layer is a runtime. A runtime is deployed remotely at the location of the data. Due to the language-agnostic representation of the analysis and release, a runtime may be written in any language, with any framework. The aim for this project is to provide one standard runtime in C++. It still remains possible to support components/algorithms that are only available in, say, Python, via a separate python runtime. In this case, the execution of an analysis/release can take multiple steps, where the analysis is partially evaluated, a release is serialized (including private data), and then passed between runtimes to continue execution.

5 Examples

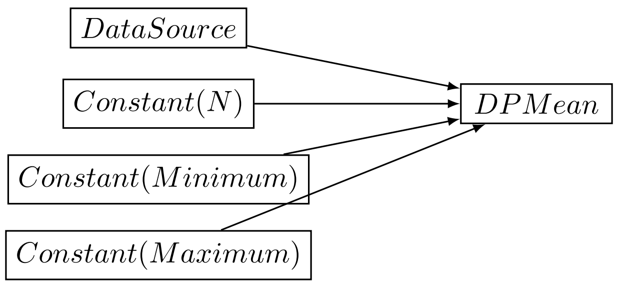
The JSON samples used in the examples are simplified.

5.1 Mean

To compute a mean, an analyst would first construct an analysis and a partial release.
1. a partial release:

```
{"N": 50, "Minimum": 23, "Maximum": 45}
```

2. an analysis:



The system passes the analysis and release to the validator and gets accuracy estimates.

```
{"DPMean": 2.3}
```

The researcher then submits the analysis to a runtime for a new release:

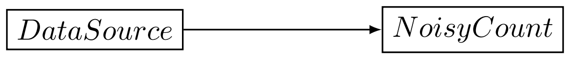
```
{"N": 500, "Minimum": 23, "Maximum": 45, "DPMean": 30.2}
```

5.2 Mean with Nesting

If the record count is private and the researcher is unable to provide an estimate, then an interactive analysis may be useful. The following metadata is now used.
1. a partial release:

```
{}
```

2. an analysis:



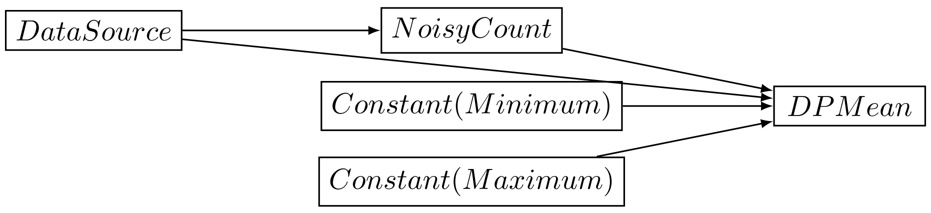
The system passes the analysis and release to the validator for accuracy estimates.

```
{"NoisyCount": 8}
```

The researcher submits the analysis to a runtime for a release.

```
{"NoisyCount": 502}
```

The researcher may now interactively extend the analysis with the mean using the estimated record count.



The system passes the analysis and release to the validator for accuracy estimates.

```
{"NoisyCount": 8, "DPMean": 2.3}
```

The researcher submits the analysis to a runtime for a release.

```
{
    "N": 50, "Minimum": 23, "Maximum": 45,
    "NoisyCount": 502, "DPMean": 30.2
}
```

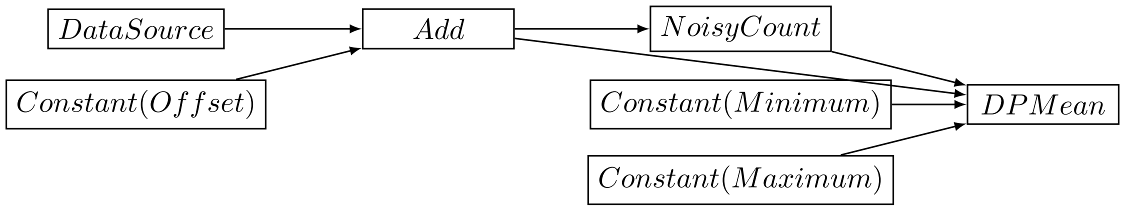
5.3 Mean with Transformation

A new analysis is started, with a separate privacy budget. Given the nature of privacy, a computational graph with released statistics is immutable- however, a graph may be extended interactively. A transformation may be applied before or after statistics. Transformations modify meta-statistics released by the validator, like accuracy and confidence. Preprocess changes to these meta-statistics are tracked via accumulated stability penalties. The following metadata is now used.

1. a partial release:

```
{"Offset": 45, "Minimum": 68, "Maximum": 90}
```

2. an analysis:



The system passes the analysis and release to the validator for accuracy estimates.

```
{"NoisyCount": 8}
```

The researcher submits the analysis to a runtime and receives an updated release.

```
{
    "Offset": 45, "Minimum": 68, "Maximum": 90,
    "NoisyCount": 498, "DPMean": 74.7
}
```

In this situation, the system was unable to provide accuracy estimates for the DPMean statistic, due to missing dependencies, but was able to release an estimate from the runtime. If accuracy estimates are desired, the release may be passed to the validator to retrieve the DPMean node's accuracy estimates.

```
{"NoisyCount": 8, "DPMean": 2.7}
```

One might expect the accuracy estimates for the DPMean node to be the same as the estimates in the previous example, because Add is a stability-1 transformation. However, this is a separate analysis from the previous example, and accuracy is dependent on the random quantity released from the NoisyCount node.

6 Conclusion

Much of the verbosity of the system can be hidden behind language bindings. The bindings can automatically initialize constant nodes, the interfaces between the validator and runtimes can be unified, and languages that support operator overloading can build up function graphs implicitly.

The flexibility of the core architecture permits language bindings with seamless construction of verifiable and secure statistical analyses.

The proposal outlines a language-agnostic centralized differential privacy library. The analysis abstraction permits bindings, parsers and runtimes in any language and on any platform, as the need arises. By using modular components, security-critical code is isolated. A single C++ runtime will allow bindings from any language to compute differentially private analyses. At the same time, potential runtimes in SQL, Spark, Dask, Pandas or C++, may implement a subset or all available primitives as desired.