A Programming Framework for the DP Library

Marco Gaboardi
(Boston University)

Michael Hay
(Colgate University)

Salil Vadhan
(Harvard University)

Timeline

September 2019 - Kickoff - Design Committee started to meet weekly to brainstorm the main requirements and desiderata. This included: Merce Crosas, James Honaker, Gary King, Aleksandra Korolova,and Ilya Mironov.

December 2019 - Design Committee retreat - Sketch of the basic design of the library and further discussions on the integration of the library with other components.

May 2020 - White paper - Outline of the different components and requirements for them. Basic prototyping of a proof of concept.

Previous work - background

Many DP systems and languages developed by research community and industry. An incomplete list: PINQ, Fuzz, Ektelo, DFuzz, wPINQ, Flex, Chorus, Airavat, Featherweight PINQ, DiffPrivLib, PSI library, PrivateSQL, APEX, Google SQL, DPella, LightDP, Duet, TensorFlow Privacy, ...

Our design borrows from prior frameworks, especially PINQ, Fuzz, Ektelo.

We welcome your feedback! **Come to breakout tomorrow 11:00 ET!**

Desiderata

- **Modularity**
- Verifiability
- Flexibility
- Extensibility
- Programmability
- Usability
- Efficiency
- Utility

Outline of talk

- Library components
  - Measurements & transformations
  - Chaining, composition, post-processing
  - Privacy and stability relations
  - Interactive measurements
- Practical considerations
  - Implementation considerations
  - Using and contributing to library

Initial assumptions (to be relaxed)

Private input **x** is of type Multiset(X)

A person may contribute ≤ 1 record to dataset

Goal: support epsilon-DP computation (“pure DP”)

(Adjacency defined in terms of symmetric difference)

Measurements and Transformations

Measurement:

A randomized function from datasets to outputs

For now, think non-interactive pure DP function

Measurement attributes
- Input domain
- Function
- Privacy loss

Example:

```
> NoisySum = MakeNoisySum(l, u, eps)

> NoisySum(private_data)
```

Expects: `private_data` is a Multiset(`[l, u]`)

Returns: sum + Laplace noise

Privacy loss: `eps`

Measurements and Transformations

Transformation:

A deterministic function from datasets to datasets.

T is c-**stable** if for any x, x' that differ by *d* records, then T(x),T(x') differ by at most *cd* records

Transformation attributes
- Input domain
- Output domain
- Function
- Stability

Example:

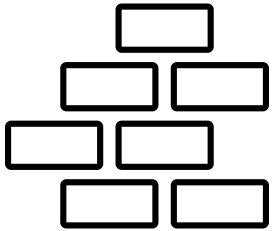
```
> Clamping = MakeClamp(l, u)

> Clamping(private_data)
```

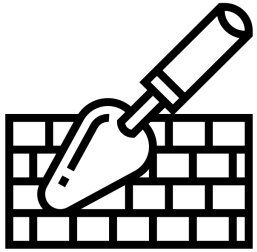
Expects: `private_data` is a Multiset(`float`)

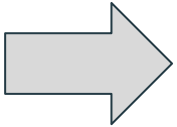
Returns: input with each item "clamped" to `[l, u]`

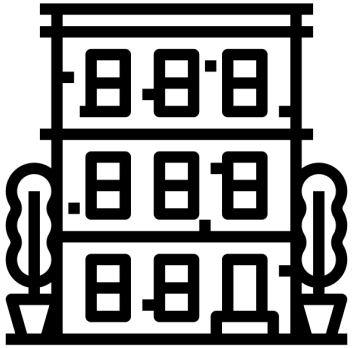
Stability: `c=1`

Combining measurements and transformations









Measurements & Transformations

Chaining, Composition, & Post-processing

Complex DP programs

Chaining

Chaining T2 and T1 yields a new transformation T’(data) := T2(T1(data))

Chaining M and T yields a new measurement: M’(data) := M(T(data))

The chaining operator

1. checks compatibility of operators
2. derives (privacy) properties of new operator

Example:

```
> NoisyClampedSum=ChainingMT(NoisySum, Clamping)
```

New measurement operator

- Input domain: multiset of floats
- Function: noisy sum of clamped values
- Privacy loss: `(c x eps)`

Chaining, Composition, Post-processing

Chaining

M’ = M(T(data))

T’ = T2(T1(data))

Post-processing

M’ = f(M(data))

Composition

M’ = (M1(data), M2(data)) sequential

M’ = (*y* = M1(data), M2(data, *y*)) adaptive

Illustrative example

Noisy average of (clamped) data

```
NoisyCount=ChainingMT(
            MakeNoisySum(1, 1, eps),
            MakeClamp(1, 1))

NoisyPair=Compose(NoisyClampedSum,
                  NoisyCount)

def divide(x,y): return x/y

NoisyMean=Postprocess(NoisyPair, divide)
```

New measurement operator

- Input domain: multiset of floats
- Privacy loss: `2eps`
- Function: noisy average of clamped values

Verifying Privacy Properties

Any contribution requires a **proof** that for all inputs,

- it raises exception or produces a *valid* measurement/transformation
- it does not modify already constructed operators or other library code

The proof can be supplied by contributor (and verified by OpenDP) or derived automatically.

Varying types and measures

Want to support multiple types of private data. Why?

1. Natural variety in input data types (tables, graphs, streams, ...)
2. Intermediate data representations (multisets, vectors, …)

Example of #2: Rewrite `NoisySum` as...

```
BoundedSum = MakeBoundedSum(l, u)
BaseLaplace = MakeBaseLap(sigma)
NoisySum=ChainingMT(BaseLaplace, BoundedSum)
```

Intermediate data type: a single float

Varying types and measures

To support multiple types, properties like stability and differential privacy must be made type-compatible

We add ***metrics*** to measurements and transformations

Example:

Output metric of `BoundedSum` and input metric of `BaseLaplace` is

$$\mathbf{d}_{\mathbf{abs}}(a, b) = |\ a - b\ |$$

Measurement attributes
- Input domain
- ***Input metric***
- Function
- Privacy loss

Transformation attributes
- Input domain
- ***Input metric***
- Output domain
- ***Output metric***
- Function
- Stability

Privacy relations and stable relations

To increase flexibility, we replace the privacy loss attribute with output measure and privacy relation. These two attributes permit to capture other notions of privacy. e.g. approx. DP, Renyi DP, or CDP.

Measurement attributes

- Input domain
- Input metric
- ***Output measure***
- Function
- ***Privacy relation***

Example:

```
> BaseGauss = MakeBaseGauss(sigma)
```

Output measure: `approxDP`

Privacy relation:

R(d_in,(eps,delta))=

(d_in/sigma)√2ln(1.25/delta) ≤ min(eps,1)

Privacy relations and stable relations

We generalize the privacy loss to output measure and privacy relation, to capture other notions of privacy. E.g. approx. DP, Renyi DP, or CDP.

Similarly, we generalize stability in transformations to stability relation. This can be used to capture more general transformations. E.g. bounded joins.

Measurement attributes

- Input domain
- Input metric
- ***Output measure***
- Function
- ***Privacy relation***

Transformation attributes

- Input domain
- Input metric
- Output domain
- Output metric
- Function
- ***Stability relation***

Chaining and composition revisited

Chaining and composition now can use the relations of the components to derive the relation of the chained transformation or measurement.

Example for chainingMT:

```
....
def privacy_relation(d_in,d_out)=
  d_mid=hint(d_in,d_out)
  return (trans.stability_relation(d_in,d_mid)
          and
          meas.privacy_relation(d_mid,d_out))
```

Measurement attributes

- Input domain
- Input metric
- ***Output measure***
- Function
- ***Privacy relation***

Transformation attributes

- Input domain
- Input metric
- Output domain
- Output metric
- Function
- ***Stability relation***

Interactive Measurements

Given the data, an interactive measurement creates a queryable object - a state machine consisting of an initial state and an evaluation function.

When the queryable receives a query, the evaluation function evaluates it and updates the state.

InteractiveMeasurement attributes

- Input domain
- Input metric
- Output measure
- Function
- Privacy relation

Queryable

- ***state***
- ***eval***

Example: Adaptive Composition

```
....
def eval(query: Measurement, state):
  (st_data, eps) = state
  if query.privacy_loss < eps:
  return (query.function(st_data),eps-query.privacy_loss)
```

Chaining, Composition, Post-processing for interactive Measurements

Chaining

M’ = M(T(data))

Post-processing

M’ = f(M(data))

f can take the queryable produced by M a produce a new queryable.

Composition

Q = M(data); a1 = Q(q1), a2 = Q(q2), … adaptive

We can also have other forms of composition, e.g. sequential, concurrent.

Outline of talk

- Library components
  - Measurements & transformations
  - Chaining, composition, post-processing
  - Privacy and stability relations
  - Interactive measurements
- Practical considerations
  - Implementation considerations
  - Using and contributing to library

Ensuring Privacy in Implementations

The design we outlined above guarantees private data to be accessed only by means of valid measurement and transformations.

In addition, we will need to prevent leakages potentially caused by:

- Timing channels
- Implementation of arithmetic
- Use of pseudorandomness

Discussion of Implementation Language

Desired features:

- memory safety, encapsulation, immutability
- abstract data types & type safety
- other: functional programming features, generics, structures

Proposal: Rust + Python and R

- Core library written in Rust (for efficiency, verifiability)
- Made available in Python and R via API bindings (for programmability)

Using the library

Calling the library from a DP system requires to:

- determine the dataset and its type, the privacy notion and its granularity.
- select an interactive measurement from the library.
- present all the queries to the queryable created by the measurement.

The library needs information about the system, e.g. data access model, capabilities of the backend, partition between secure and insecure storage.

Different user interfaces can be built on top of the library, e.g. sql-like, GUI, Python notebook, etc.

Contributing to the library

We envision different kinds of code contributions:

- New measurements or transformations combining existing library components.
- New private data types, distance measures, or privacy notions.
- New primitives to combine measurements and transformations.
- New “atomic” measurements or transformations with proof of correctness.
- New type of privacy or stability calculus.

The Scope of the Framework

What is supported within the current framework:

- Many different dataset types, privacy measures and granularities, ways to combine different primitives to build more complex mechanisms.
- Common database transformations, mechanisms based on global sensitivity or restricted sensitivity

Outside the current framework:

- Mechanisms based on local sensitivity, privacy odometers, privacy with explicit adversary models, randomized or interactive transformations.

Summary

- Library components
  - Measurements & transformations
  - Chaining, composition, post-processing
  - Privacy and stability relations
  - Interactive measurements
- Practical considerations
  - Implementation considerations
  - Using and contributing to library