Privacy Integrated Data Stream Queries

Lucas Waye

Workshop on Privacy and Security in Programming 2014
Sanitizing Private Data Sets
Sanitizing Private Data Sets

Private Static Data Set
Sanitizing Private Data Sets

Static Data Set

Sanitizer
Differentially Private

Private

Static Data Set
Sanitizing Private Data Sets

Private
Static Data Set

Sanitizer
Differentially Private

Public
Sanitized Output
Sanitizing Private Data Sets

Private
Static Data Set

Sanitizer
Differentially Private

Public
Sanitized Output
Sanitizing Private Data Sets

Private
Static Data Set
- Row database
- Graph
- Bids

Sanitizer
Differentially Private

Public
Sanitized Output
Sanitizing Private Data Sets

- **Static Data Set**: Row database, Graph, Bids
- **Sanitizer**: Differentially Private
- **Sanitized Output**: Query results, Synthetic data, Summary Structure

Private → Sanitizer → Public
Sanitizing Private Data Sets

Available to non-privacy-experts

Private Static Data Set
- Row database
- Graph
- Bids

Sanitizer
Differentially Private

Sanitized Output
- Query results
- Synthetic data
- Summary Structure

Public
Streaming Data
Streaming Data
Streaming Data
Streaming Data
Streaming Data
Streaming Data

traditional differentially private mechanisms do not account for new data ("one-shot")

streaming sanitizers not accessible to non-privacy-experts
Streaming Data

this talk — bringing theory to practice
giving non-experts the ability to sanitize private streaming data

traditional differentially private mechanisms
do not account for new data ("one-shot")

streaming sanitizers not accessible to non-privacy-experts
Talk Outline

• Background: streaming differential privacy
  • Event-Level privacy
  • User-Level Privacy

• Our setting

• Streaming PINQ
  • Where PINQ falls short
  • Streaming PINQ agents by example

• Conclusions and future work
Differentially Private Streaming Algorithms

Private Streaming Input

Streaming Sanitizer

Public Streaming Output
Differentially Private Streaming Algorithms

- Private Streaming Input
- Streaming Sanitizer
- Public Streaming Output
Privacy Guarantees

it varies!
Privacy Guarantees

it varies!
Privacy Guarantees

it varies!

Based on theoretical output behavior:
The output of the sanitizer does not differ much on neighboring input streams.
Privacy Guarantees

it varies!

Based on theoretical output behavior:
The output of the sanitizer does not differ much on neighboring input streams.

Result: hard to notice if a particular individual is present in the data set
Privacy Guarantees

it varies!

Based on theoretical output behavior:
The output of the sanitizer does not differ much on neighboring input streams.

Result: hard to notice if a particular individual is present in the data set

How much does the output differ?
Privacy Guarantees

it varies!

Based on theoretical output behavior:
The output of the sanitizer does not differ *much* on neighboring input streams.

**Result**: hard to notice if a particular individual is present in the data set

How *much* does the output differ?

What is a *neighboring* input stream?
- event-level privacy
- user-level privacy
Event-Level Privacy*

Original Event Stream

neighbors differ by only one event

colors represent different event types

* Dwork et al. 2010
Event-Level Privacy*

Original Event Stream

colors represent different event types

neighbors differ by only one event

* Dwork et al. 2010
Event-Level Privacy*

Original Event Stream

neighbors differ by only one event

* Dwork et al. 2010

colors represent different event types
User-Level Privacy*

colors represent different users’ events

neighbors differ by only one user

* Dwork et al 2010
User-Level Privacy*

Original Event Stream

colors represent different users’ events

neighbors differ by only one user

first user differs

* Dwork et al 2010
User-Level Privacy*
Talk Outline

• Background: streaming differential privacy
  • Event-Level privacy
  • User-Level Privacy

• Our setting

• Streaming PINQ
  • Where PINQ falls short
  • Streaming PINQ agents by example

• Conclusions and future work
Our Setting

Data Owner

Analyst

Private

Streaming Input
Our Setting

Data Owner

Analyst

useful information

protect privacy

Private Streaming Input
Our Setting

Data Owner

useful information

protect privacy

Analyst

Private Streaming Input

User-Level Private

Event-Level Private

Event-Level Private

Streaming Sanitizers with different privacy guarantees
Our Setting

Data Owner

Agent enforces privacy requirements of data owner

useful information

protect privacy

Analyst

User-Level Private

Event-Level Private

Streaming Sanitizers with different privacy guarantees

Private Streaming Input
Our Setting

Data Owner

Analyst

Agent enforces privacy requirements of data owner

useful information

protect privacy

selects sanitizers to use

User-Level Private
Event-Level Private

Streaming Sanitizers with different privacy guarantees

Private
Streaming Input
Our Setting

Data Owner

Analyst

useful information
protect privacy
selects sanitizers to use
informs agent of privacy properties
Agent enforces privacy requirements of data owner
User-Level Private
Event-Level Private
Event-Level Private
Streaming Sanitizers with different privacy guarantees
Streaming Input

Private

Agent enforces privacy requirements of data owner
selects sanitizers to use
informs agent of privacy properties

Useful information
protect privacy

Privacy properties

Our Setting

Data Owner

Analyst

useful information
protect privacy

Agent enforces privacy requirements of data owner

User-Level Private
Event-Level Private
Event-Level Private

Streaming Sanitizers with different privacy guarantees

Private
Streaming Input
Our Setting

Data Owner

Analyst

useful information

protect privacy

Agent enforces privacy requirements of data owner

Private Streaming Input

public event

private event

private event

Streaming Sanitizers with different privacy guarantees

User-Level Private

Event-Level Private

Event-Level Private
Talk Outline

• Background: streaming differential privacy
  • Event-Level privacy
  • User-Level Privacy

• Our setting

• Streaming PINQ
  • Where PINQ falls short  see paper for how other related work falls short
  • Streaming PINQ agents by example

• Conclusions and future work
var tweets = ReadAllSavedTweets("saved_tweets.txt");
var agent = new PINQAgentBudget(1.0);
var data = new PINQueryable<Tweet>(tweets, agent);

double tweetsFromNY = data
    .Where(tweet => tweet.Location.State == "NY")
    .NoisyCount(1.0);

Console.WriteLine("Tweets from New York: " + tweetsFromNY);

* McSherry 2009
PINQ

Privacy Integrated Query*

-var tweets = ReadAllSavedTweets("saved_tweets.txt");
-var agent = new PINQAgentBudget(1.0);
-var data = new PINQueryable<Tweet>(tweets, agent);

double tweetsFromNY = data.Where(tweet => tweet.Location.State == "NY") .NoisyCount(1.0);

Console.WriteLine("Tweets from New York: " + tweetsFromNY);

* McSherry 2009
PINQ
Privacy Integrated Query*

Var tweets = ReadAllSavedTweets("saved_tweets.txt");
var agent = new PINQAgentBudget(1.0);
var data = new PINQueryable<Tweet>(tweets, agent);

double tweetsFromNY = data
    .Where(tweet => tweet.Location.State == "NY")
    .NoisyCount(1.0);

Console.WriteLine("Tweets from New York: " + tweetsFromNY);

* McSherry 2009
var tweets = ReadAllSavedTweets("saved_tweets.txt");
var agent = new PINQAgentBudget(1.0);
var data = new PINQueryable<Tweet>(tweets, agent);

double tweetsFromNY = data
    .Where(tweet => tweet.Location.State == "NY")
    .NoisyCount(1.0);

Console.WriteLine("Tweets from New York: " + tweetsFromNY);
PINQ: Streaming?

```csharp
var tweets = ReadAllSavedTweets("saved_tweets.txt");
var agent = new PINQAgentBudget(1.0);
var data = new PINQueryable<Tweet>(tweets, agent);

double tweetsFromNY = data
    .Where(tweet => tweet.Location.State == "NY")
    .NoisyCount(1.0);

Console.WriteLine("Tweets from New York: " + tweetsFromNY);
```
var tweets = ReadAllSavedTweets("saved_tweets.txt");
var agent = new PINQAgentBudget(1.0);
var data = new PINQueryable<Tweet>(tweets, agent);

double tweetsFromNY = data
  .Where(tweet => tweet.Location.State == "NY")
  .NoisyCount(1.0);

Console.WriteLine("Tweets from New York: " + tweetsFromNY);
PINQ: Streaming?

```csharp
var tweets = ReadAllSavedTweets("saved_tweets.txt");
var agent = new PINQAgentBudget(1.0);
var data = new PINQueryable<Tweet>(tweets, agent);

double tweetsFromNY = data
    .Where(tweet => tweet.Location.State == "NY")
    .NoisyCount(1.0);

Console.WriteLine("Tweets from New York: "+ tweetsFromNY);
```

static data set

user-level or event-level?
PINQ: Streaming?

```csharp
var tweets = ReadAllSavedTweets("saved_tweets.txt");
var agent = new PINQAgentBudget(1.0);
var data = new PINQueryable<Tweet>(tweets, agent);

double tweetsFromNY = data
    .Where(tweet => tweet.Location.State == "NY")
    .NoisyCount(1.0);

Console.WriteLine("Tweets from New York: " + tweetsFromNY);
```

- static data set
- user-level or event-level?
- get result immediately
PINQ: Streaming?

```csharp
var tweets = ReadAllSavedTweets("saved_tweets.txt");
var agent = new PINQAgentBudget(1.0);
var data = new PINQueryable<Tweet>(tweets, agent);

double tweetsFromNY = data
    .Where(tweet => tweet.Location.State == "NY")
    .NoisyCount(1.0);

Console.WriteLine("Tweets from New York: " + tweetsFromNY);
```

Contributions

- Support for streaming events
- New agents that are aware of streaming privacy properties
- Five differentially private streaming algorithm implementations
Streaming PINQ

```csharp
var tweets = AllTweetsFireHose(); // custom data provider
var agent = new EventLevelPrivacyBudget(1.0); // streaming agent
var data = new StreamingQueryable<Tweet>(tweets, agent);

// returns handle to output stream
double tweetsFromNY = data.Where(tweet => tweet.Location.State == "NY")
    .RandomizedResponseCount(1.0);

// callback when output is made by algorithm
tweetsFromNY.OnOutput = (c =>
    Console.WriteLine("Tweets from New York: " + c));

// process 5,000 events
tweetsFromNY.ProcessEvents(5000);
```

See paper for:
• description of streaming event API
• implemented streaming algorithms
var tweets = AllTweetsFireHose(); // custom data provider
var agent = new EventLevelPrivacyBudget(1.0); // streaming agent
var data = new StreamingQueryable<Tweet>(tweets, agent);

// returns handle to output stream
double tweetsFromNY = data
    .Where(tweet => tweet.Location.State == "NY")
    .RandomizedResponseCount(1.0);

// callback when output is made by algorithm
tweetsFromNY.OnOutput = (c =>
    Console.WriteLine("Tweets from New York: " + c));

// process 5,000 events
tweetsFromNY.ProcessEvents(5000);

See paper for:
• description of streaming event API
• implemented streaming algorithms
Mixing Privacy Guarantees

- User-Level Private
- Event-Level Private

Event-Level Agent
Budget = $2\varepsilon$

ε privacy
Mixing Privacy Guarantees

User-Level Private

Event-Level Private

ε privacy

ε privacy

Event-Level Agent

Budget = 2ε

time
Mixing Privacy Guarantees

- User-Level Private
- Event-Level Private
- Event-Level Agent
- \( \varepsilon \) privacy

Budget = 2\( \varepsilon \)
Mixing Privacy Guarantees

User-Level Private

Event-Level Private

Event-Level Agent
Budget = $2\varepsilon$

$\varepsilon$ privacy

$\varepsilon$ privacy
Mixing Privacy Guarantees

User-Level Private

Event-Level Private

Event-Level Agent
Budget = 2ε

ε privacy
Mixing Privacy Guarantees

Event-Level Agent
Budget = 2\(\varepsilon\)

User-Level Private
\(\varepsilon\) privacy

Event-Level Private
\(\varepsilon\) privacy
Mixing Privacy Guarantees

User-Level Privacy implies Event-Level Privacy

Event-Level Agent
Budget = 2\(\varepsilon\)

User-Level Private
\(\varepsilon\) privacy

Event-Level Private
\(\varepsilon\) privacy
Mixing Privacy Guarantees

User-Level Agent
Budget = $2\varepsilon$

User-Level Private
$\varepsilon$ privacy

Event-Level Private
$\varepsilon$ privacy
Mixing Privacy Guarantees

User-Level Private

ε privacy

Event-Level Private

ε privacy

User-Level Agent

Budget = 2ε
Mixing Privacy Guarantees

User-Level Private

Event-Level Private

User-Level Agent
Budget = $2\varepsilon$

$\varepsilon$ privacy
Mixing Privacy Guarantees

User-Level Private

Event-Level Private

User-Level Agent
Budget = 2ε

privacy of stream permanently affected
event-level treated as a one-time use of user-level algorithm

ε privacy
Mixing Privacy Guarantees

User-Level Private

Event-Level Private

User-Level Agent

Budget Exhausted

privacy of stream permanently affected

event-level treated as a one-time use of user-level algorithm
Mixing Privacy Guarantees

- User-Level Private
- Event-Level Private

User-Level Agent
Budget Exhausted

Privacy of stream permanently affected
event-level treated as a one-time use of user-level algorithm

ε privacy

ε privacy

ε privacy
Mixing Privacy Guarantees

User-Level Private

Event-Level Private

User-Level Agent

Budget Exhausted

privacy of stream permanently affected

event-level treated as a one-time use of user-level algorithm

privacy

ε privacy

ε privacy
Mixing Privacy Guarantees

User-Level Private

User-Level Agent

Budget Exhausted

Privacy of stream permanently affected

event-level treated as a one-time use of user-level algorithm

Event-Level Private

ε privacy

ε privacy

User-Level Private

ε privacy
Mixing Privacy Guarantees

User-Level Private

Event-Level Private

User-Level Agent

Budget = 2

\( \varepsilon \) privacy

Privacy of stream permanently affected

Event-level treated as a one-time use of user-level algorithm

See paper for full description of streaming agent API

\( \varepsilon \) privacy
Future Work

• Including timing of events in the model
  • Stock trade made after hours \(\rightarrow\) institutional trader
  • Time boxing? Incorporate research on timing channels?

• Large trusted code base
  • Programming framework provides no help in assuring new streaming algorithm implementations are safe
  • C# seems to be the wrong choice of language: many side-channels
  • DSL for streaming queries — what are the right primitives?
Questions?

Thanks!