What are we trying to solve?

Why Differential Privacy?

What is Differential Privacy?

What is PINQ?

What functionality does PINQ offer?

How does PINQ operate?
• Select 5 people at random from the room for a health survey
• Release aggregate statistics
- Select 5 people at random from the room for a health survey
- Release aggregate statistics
• Select 5 people at random from the room for a health survey
• Release aggregate statistics
• Select 5 people at random from the room for a health survey
• Release aggregate statistics

AVERAGE AGE

A: 6Y 5M
• Select 5 people at random from the room for a health survey
• Release aggregate statistics

AVERAGE AGE

A: 6Y 5 M
B: 12Y 2 M
• Select 5 people at random from the room for a health survey
• Release aggregate statistics
- Select 5 people at random from the room for a health survey
- Release aggregate statistics

AVERAGE AGE

A: 6 Y 5 M
B: 12 Y 2 M
Given that we need to reveal something about the dataset…
Given that we need to reveal something about the dataset…
Given that we need to reveal something about the dataset…

ACCURACY

PRIVACY
Given that we need to reveal something about the dataset…
Given that we need to reveal something about the dataset…
Given that we need to reveal something about the dataset…

- Want ‘reasonable’ numbers
- Be able to give some ‘privacy guarantee’ to a user – no deanonymization
AVERAGE AGE

A: 6Y 5 M

B: 12Y 2 M
AVERAGE AGE

A: 6 Y 5 M

B: 12 Y 2 M

A ~ 10 Y 3 M
AVERAGE AGE

A ~ 10 Y 3 M
B ~ 11 Y 1 M
AVERAGE AGE

A ~ 10Y 3M

B ~ 11Y 1M
What guarantee can you give a user?

A ~ 10Y 3M
B ~ 11Y 1M
What guarantee can you give a user?

The published answer looks ‘roughly’ the same if the user is present or absent.

A ~ 10Y 3M

B ~ 11Y 1M
What guarantee can you give a user?

The published answer looks ‘roughly’ the same if the user is present or absent.
DEFINITION (DWORK ET AL.)
A randomized computation $M$ provides $\epsilon$-differential privacy if for any two data sets $A$ and $B$, and any set of possible outputs $S \subseteq Range(M)$,
A randomized computation $M$ provides $\epsilon$-differential privacy if for any two data sets $A$ and $B$, and any set of possible outputs $S \subseteq Range(M)$,

$$\Pr[M(A) \in S] \leq \Pr[M(B) \in S] \times \exp(\epsilon \times |A \oplus B|)$$
DEFINITION (DWORK ET AL.)

A randomized computation $M$ provides $\epsilon$-differential privacy if for any two data sets $A$ and $B$, and any set of possible outputs $S \subseteq \text{Range}(M)$,

$$\Pr[M(A) \in S] \leq \Pr[M(B) \in S] \times \exp(\epsilon \times |A \oplus B|)$$
A randomized computation $M$ provides $\epsilon$-differential privacy if for any two data sets $A$ and $B$, and any set of possible outputs $S \subseteq \text{Range}(M)$,

$$\Pr[M(A) \in S] \leq \Pr[M(B) \in S] \times \exp(\epsilon \times |A \oplus B|)$$
A randomized computation $M$ provides $\epsilon$-differential privacy if for any two data sets $A$ and $B$, and any set of possible outputs $S \subseteq Range(M)$,

$$\Pr[M(A) \in S] \leq \Pr[M(B) \in S] \times \exp(\epsilon \times |A \oplus B|)$$
A randomized computation $M$ provides $\epsilon$-differential privacy if for any two data sets $A$ and $B$, and any set of possible outputs $S \subseteq \text{Range}(M)$,

$$\Pr[M(A) \in S] \leq \Pr[M(B) \in S] \times \exp(\epsilon \times |A \oplus B|)$$
A randomized computation $M$ provides $\epsilon$-differential privacy if for any two data sets $A$ and $B$, and any set of possible outputs $S \subseteq Range(M)$,

$$\Pr[M(A) \in S] \leq \Pr[M(B) \in S] \times \exp(\epsilon \times |A \oplus B|)$$

~ Not too different
A randomized computation $M$ provides $\epsilon$-differential privacy if for any two data sets $A$ and $B$, and any set of possible outputs $S \subseteq Range(M)$,

$$\Pr[M(A) \in S] \leq \Pr[M(B) \in S] \times \exp(\epsilon \times |A \oplus B|)$$

- $A \sim 10, 3M$
- $B \sim 11, 1M$
- $S = [10, 11]$
- Add random noise to Avg Age
- $\sim$ Not too different

For one user = 1
A randomized computation $M$ provides $\epsilon$-differential privacy if for any two data sets $A$ and $B$, and any set of possible outputs $S \subseteq Range(M)$,

$$\Pr[M(A) \in S] \leq \Pr[M(B) \in S] \times \exp(\epsilon \times |A \oplus B|)$$

- $A \sim \{10\} \cup \{3M\}$
- $B \sim \{11\} \cup \{1M\}$
- $S = [10, 11]$ [Add random noise to Avg Age]

- ~ Not too different
- Depends on $\epsilon$!
- For one user $= 1$
A randomized computation $M$ provides $\epsilon$-differential privacy if for any two data sets $A$ and $B$, and any set of possible outputs $S \subseteq Range(M)$,

$$\Pr[M(A) \in S] \leq \Pr[M(B) \in S] \times \exp(\epsilon \times |A \oplus B|)$$

Add random noise to Avg Age

$S = [10, 11]$

What happens as $\epsilon$ becomes greater?

~ Not too different Depends on $\epsilon$!
BUT...
BUT... Can we add ANY noise?
BUT...

Can we add **ANY** noise?

Can we allow **ANY** number of queries?
BUT...

Can we add **ANY** noise?

Can we allow **ANY** number of queries?
BUT...

Can we add ANY noise?

Can we allow ANY number of queries?
ENTER PINQ (PRIVACY INTEGRATED QUERIES)
ENTER PINQ *(PRIVACY INTEGRATED QUERIES)*

- Declarative programming platform based on LINQ (Language Integrated Queries) in C#
ENTER PINQ (PRIVACY INTEGRATED QUERIES)

• Declarative programming platform based on LINQ (Language Integrated Queries) in C#
• SQL-like syntax to interact with data source
ENTER PINQ (PRIVACY INTEGRATED QUERIES)

- Declarative programming platform based on LINQ (Language Integrated Queries) in C#
- SQL-like syntax to interact with data source

Example 1: Counting searches from distinct users in PINQ.

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

var users = from record in data
             where record.Query == argv[0]
             group by record.IPAddress
             select record.

Console.WriteLine(argv[0] + ": " + users.NoisyCount(0.1));
```
ENTER PINQ (PRIVACY INTEGRATED QUERIES)

• Declarative programming platform based on LINQ (Language Integrated Queries) in C#
• SQL-like syntax to interact with data source
ENTER PINQ (PRIVACY INTEGRATED QUERIES)

• Declarative programming platform based on LINQ (Language Integrated Queries) in C#
• SQL-like syntax to interact with data source
• Make differential privacy accessible
ENTER PINQ (PRIVACY INTEGRATED QUERIES)

• Declarative programming platform based on LINQ (Language Integrated Queries) in C#
• SQL-like syntax to interact with data source
• Make differential privacy accessible
• Data owners protect data sources with a differential privacy ‘budget’
**ENTER PINQ** *(PRIVACY INTEGRATED QUERIES)*

- Declarative programming platform based on LINQ (Language Integrated Queries) in C#
- SQL-like syntax to interact with data source
- Make differential privacy accessible
- Data owners protect data sources with a differential privacy ‘budget’
- PINQ manages this privacy budget
**ENTER PINQ** *(PRIVACY INTEGRATED QUERIES)*

- Declarative programming platform based on LINQ (Language Integrated Queries) in C#
- SQL-like syntax to interact with data source
- Make differential privacy accessible
- Data owners protect data sources with a differential privacy ‘budget’
- PINQ manages this privacy budget
- Privacy guarantees come from differential privacy
PINQ SYSTEM

- Central Type – PINQueryable
- PINQueryable = IQueryable (unprotected) + PINQAgent (privacy)
TRACKING PRIVACY
TRACKING PRIVACY

• How does a PINQAgent calculate the privacy cost for every query?
• How does a PINQAgent calculate the privacy cost for every query?
• What kinds of queries are supported?
TRACKING PRIVACY

- How does a PINQAgent calculate the privacy cost for every query?
- What kinds of queries are supported?

Aggregations:
COUNT
SUM
MEDIAN
AVG
TRACKING PRIVACY

• How does a PINQAgent calculate the privacy cost for every query?
• What kinds of queries are supported?

Aggregations:
COUNT
SUM
MEDIAN
AVG

Transformations:
WHERE
SELECT
GROUP BY
JOIN*
TRACKING PRIVACY

• How does a PINQAgent calculate the privacy cost for every query?
• What kinds of queries are supported?

Aggregations:
COUNT
SUM
MEDIAN
AVG

Transformations:
WHERE
SELECT
GROUP BY
JOIN*
NOISY COUNT
NOISY COUNT

Count = 107 + Laplace noise – would only shift by a multiplicative factor if Count = 108 instead – if the Laplace noise parameter is $1/\epsilon$ you get $\epsilon$-differentially private counts
NOISY COUNT

Count = 107 + Laplace noise – would only shift by a multiplicative factor if Count = 108 instead – if the Laplace noise parameter is $1/\epsilon$ you get differentially private counts.

What happens as $\epsilon$ becomes greater?
NOISY COUNT

Count = 107 + Laplace noise – would only shift by a multiplicative factor if Count = 108 instead – if the Laplace noise parameter is $1/\epsilon$ you get differentially private counts.

What happens as $\epsilon$ becomes greater?

Is this enough?
C-STABLE TRANSFORMATIONS
C-STABLE TRANSFORMATIONS

When you apply the transformation with and without a user, how many records in the output change?
C-STABLE TRANSFORMATIONS

When you apply the transformation with and without a user, how many records in the output change?

WHERE
SELECT
GROUP BY
JOIN*
C-STABLE TRANSFORMATIONS

When you apply the transformation with and without a user, how many records in the output change?

```
WHERE
SELECT
GROUP BY
JOIN*
C = 1
```
C-STABLE TRANSFORMATIONS

When you apply the transformation with and without a user, how many records in the output change?

```
WHERE C = 1
SELECT C = 1
GROUP BY
JOIN*
```
C-STABLE TRANSFORMATIONS

When you apply the transformation with and without a user, how many records in the output change?

WHERE
SELECT
GROUP BY
JOIN*

C = 1
C = 1
C = 2
C-STABLE TRANSFORMATIONS

When you apply the transformation with and without a user, how many records in the output change?

WHERE C = 1
SELECT C = 1
GROUP BY C = 2
JOIN* C = 2
C-STABLE TRANSFORMATIONS

When you apply the transformation with and without a user, how many records in the output change?

WHERE
SELECT
GROUP BY
JOIN*

C = 1
C = 1
C = 2
C = 2

An $\epsilon$-DP aggregation on a c-stable transformation gives $c\epsilon$-DP
HOW DOES IT WORK?
Example 5 Measuring query frequencies in PINQ.

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

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

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

Example 5 Measuring query frequencies in PINQ.

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

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

// output the count to the screen, or anywhere else
Console.WriteLine(args[0] + " : " + users.NoisyCount(0.1));
```
Example 5 Measuring query frequencies in PINQ.

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

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

// output the count to the screen, or anywhere else
Console.WriteLine(args[0] + "\":\n" + users.NoisyCount(0.1));
```
Example 5 Measuring query frequencies in PINQ.

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

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

// output the count to the screen, or anywhere else
Console.WriteLine(args[0] + " : " + users.NoisyCount(0.1));
Example 5 Measuring query frequencies in PINQ.

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

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

// output the count to the screen, or anywhere else
Console.WriteLine(args[0] + "": " + users.NoisyCount(0.1));
```
Example 5 Measuring query frequencies in PINQ.

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

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

// output the count to the screen, or anywhere else
Console.WriteLine(args[0] + ": " + users.NoisyCount(0.1));
```
MULTIPLE QUERIES (COMPOSITION)
Sequential $\epsilon$-DP analyses simply ADD $\epsilon$ for total effective $\epsilon$.
Sequential $\epsilon$-DP analyses simply ADD $\epsilon$ for total effective $\epsilon$
MULTIPLE QUERIES (COMPOSITION)

Sequential $\epsilon$-DP analyses simply ADD $\epsilon$ for total effective $\epsilon$
Sequential $\epsilon$-DP analyses simply ADD $\epsilon$ for total effective $\epsilon$
Sequential $\epsilon$-DP analyses simply ADD $\epsilon$ for total effective $\epsilon$

Total allowed $\epsilon$
Sequential $\varepsilon$-DP analyses simply ADD $\varepsilon$ for total effective $\varepsilon$

How do you choose $\varepsilon$?
Sequential $\epsilon$-DP analyses simply ADD $\epsilon$ for total effective $\epsilon$.

\[ \exp(\epsilon \times |A \oplus B|) \approx \exp(x \ll 1) \approx 1 + x \approx 1 \]
Sequential $\epsilon$-DP analyses simply add $\epsilon$ for total effective $\epsilon$.

$$\exp(\epsilon \times |A \oplus B|) \approx \exp(x \ll 1) \approx 1 + x \approx 1$$

What $|A \oplus B|$ are you protecting?
PARTITION (PARALLEL COMPOSITION)
Parallel (Disjoint) $\epsilon$-DP analyses take $\text{MAX } \epsilon$ for total effective $\epsilon$.
Parallel (Disjoint) $\epsilon$-DP analyses take $\text{MAX}\ \epsilon$ for total effective $\epsilon$
Parallel (Disjoint) $\epsilon$-DP analyses take MAX $\epsilon$ for total effective $\epsilon$

Example 5 Measuring query frequencies in PINQ.

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

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

// output the count to the screen, or anywhere else
Console.WriteLine(args[0] + "": " + users.NoisyCount(0.1));
```
Parallel (Disjoint) $\epsilon$-DP analyses take $\text{MAX } \epsilon$ for total effective $\epsilon$

Example 5 Measuring query frequencies in PINQ.

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

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

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

What if you needed the counts for 10 queries?
Parallel (Disjoint) $\epsilon$-DP analyses take $\text{MAX} \ \epsilon$ for total effective $\epsilon$

Example 5 Measuring query frequencies in PINQ.

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

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

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

What if you needed the counts for 10 queries?

Use PARTITION
Parallel (Disjoint) $\epsilon$-DP analyses take $\text{MAX } \epsilon$ for total effective $\epsilon$

Example 5 Measuring query frequencies in PINQ.
```csharp
// prepare data with privacy budget
var agent = new PINQAgentBudget(1.0);
var data = new PINQueryable<string>(rawdata, agent);

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

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

Example 6 Measuring many query frequencies in PINQ.
```csharp
// prepare data with privacy budget
var agent = new PINQAgentBudget(1.0);
var data = new PINQueryable<string>(rawdata, agent);

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

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

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

    // output the counts to the screen, or anywhere else
    Console.WriteLine(query + ":");
    foreach (var count in new int[] {1,2,3,4,5})
        Console.WriteLine(freqs[count].NoisyCount(0.1));
}
Parallel (Disjoint) $\epsilon$-DP analyses take $\text{MAX} \epsilon$ for total effective $\epsilon$.

Example 5 Measuring query frequencies in PINQ.

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

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

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

Example 6 Measuring many query frequencies in PINQ.

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

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

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

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

  // output the counts to the screen, or anywhere else
  Console.WriteLine(query + ":" +
                    freqs.Count(count => count[0].NoisyCount(0.1));

  Console.WriteLine(freqs.Count(count => count[0].NoisyCount(0.1));
}
FINALLY...
FINALLY...

Purify

Diagram:

- Question mark connected to data stacks labeled D, A.
- Arrow pointing to cloud labeled policy.
- Cloud labeled policy connected to data stacks D, A.
FINALLY…

Performance?
DISCUSSION

• Accessible
• Not much overhead
• Mathematical guarantee
• Allows richer queries
DISCUSSION

• Accessible
• Not much overhead
• Mathematical guarantee
• Allows richer queries

• Is it really?
• Budget is finite (DP)
• One user can use up
• Data source available always
• Note: Works with static data
• PINQ 1 https://dl.acm.org/citation.cfm?id=1559850
• PINQ talk – Frank McSherry - https://www.youtube.com/watch?v=GnIB7KJ5kVg
• https://www.refinery29.com/en-us/2017/10/179039/this-is-fine-meme-halloween-costume
• https://www.vulture.com/2019/06/this-is-fine-dog-meme-comic-kc-green-interview.html
• https://knowyourmeme.com/memes/math-lady-confused-lady