Prio: Private, Robust, and Efficient Computation of Aggregate Statistics

Henry Corrigan-Gibbs and Dan Boneh
Stanford University

NSDI 2017
Today: Non-private aggregation
Today: Non-private aggregation

Each user has a private data point.
Today: Non-private aggregation

Blood pressure vs. Twitter usage scatter plot.
Today: Non-private aggregation

Blood pressure vs Twitter usage diagram
Today: Non-private aggregation

\[ B(T) = c_1 \cdot T + c_0 \]
Today: Non-private aggregation

The app provider learned more than it needed.

\[ B(T) = c_1 \cdot T + c_0 \]
Today: Non-private aggregation

Twitter usage vs. Blood pressure
This paper:
Private aggregation

Blood pressure vs Twitter usage
Clients send an *encrypted share* of their data to each aggregator.
This paper: Private aggregation

Clients send an encrypted share of their data to each aggregator
This paper:
Private aggregation

Clients send an encrypted share of their data to each aggregator.

Blood pressure

Twitter usage
This paper: Private aggregation

The aggregators learn no private client data

Blood pressure

Twitter usage

\[ B(T) = c_1 \cdot T + c_0 \]

The aggregators learn no private client data
1. **Exact correctness**  

   If all servers are honest, servers learn \( f(\cdot) \)

2. **Privacy**  

   If one server is honest, servers learn only* \( f(\cdot) \)

3. **Robustness**  

   Malicious clients have bounded influence

4. **Efficiency**  

   No public-key crypto (apart from TLS)  
   1000s of submissions per second
### Private aggregation

1. **Exact correctness**
   If all servers are honest, servers learn $f(\cdot)$

2. **Privacy**
   If one server is honest, servers learn only* $f(\cdot)$

3. **Robustness**
   Malicious clients have bounded influence

4. **Efficiency**
   No public-key crypto (apart from TLS)
   1000s of submissions per second

---

**Prio** is the first system to achieve all four.
Private aggregation

1. **Exact correctness**  
   If all servers are honest, servers learn $f(x_1, \ldots, x_N)$

2. **Privacy**  
   If one server is honest, servers learn only* $f(x_1, \ldots, x_N)$

3. **Robustness**  
   Malicious clients have bounded influence

4. **Efficiency**  
   No public-key crypto (apart from TLS)  
   1000s of submissions per second

…and Prio supports a wide range of aggregation functions $f(\cdot)$

Prio is the first system to achieve all four.
Private aggregation

\[ f(x_1, \ldots, x_N) \]

1. **Exact correctness** If all servers are honest, servers learn \( f(\cdot) \)
2. **Privacy** If one server is honest, servers learn only* \( f(\cdot) \)
3. **Robustness** Malicious clients have bounded influence
4. **Efficiency** No public-key crypto (apart from TLS)
   1000s of submissions per second

**Prio is the first system to achieve all four.**
Contributions

1. **Secret-shared non-interactive proofs (SNIPs)**
   - Client proves that its encoded submission is well-formed
   - We do not need the power of traditional “heavy” crypto tools

2. **Aggregatable encodings**
   Can compute sums privately \implies\ Can compute $f(\cdot)$ privately
   \ldots for many $f$’s of interest
Related systems

- **Additively homomorphic encryption**

- **Multi-party computation** [GMW87], [BGW88]

- **Anonymous credentials/tokens**
  VPriv (2009), PrivStats (2011), ANONIZE (2014), …

- **Randomized response** [W65], [DMNS06], [D06]
  RAPPOR (2014, 2016)

Prio is the first system to achieve exact correctness, privacy, robustness, efficiency.
Outline

• Background: The private aggregation problem
• A straw-man solution for private sums
• Providing robustness with SNIPs
• Evaluation
• Encodings for complex aggregates
Outline

• Background: The private aggregation problem
• A straw-man solution for private sums
• Providing robustness with SNIPs
• Evaluation
• Encodings for complex aggregates
Warm-up: Computing private sums
Warm-up: Computing private sums

• Every device $i$ holds a value $x_i$
• We want to compute
  
  \[ f(x_1, \ldots, x_N) = x_1 + \ldots + x_N \]

  without learning any users’ private value $x_i$. 
Warm-up: Computing private sums

- Every device $i$ holds a value $x_i$
- We want to compute
  \[ f(x_1, \ldots, x_N) = x_1 + \ldots + x_N \]
  without learning any users’ private value $x_i$.

**Example:** Privately measuring traffic congestion.

\[ x_i = 1 \quad \text{if user } i \text{ is on the Bay Bridge} \]
\[ = 0 \quad \text{otherwise} \]

The sum $x_1 + \ldots + x_N$ yields the number of app users on the Bay Bridge.
Private sums: A “straw-man” scheme

[Chaum88], [BGW88], …
[KDK11] [DFKZ13] [PrivEx14] …
Private sums: A “straw-man” scheme

Assume that the servers are non-colluding.
Equivalently: that at least one server is honest.
Private sums: A “straw-man” scheme

Assume that the servers are non-colluding. Equivalently: that at least one server is honest.
Private sums: A “straw-man” scheme

Assume that the servers are non-colluding.
Equivalently: that at least one server is honest.

[Chaum88], [BGW88], …
[KDK11] [DFKZ13] [PrivEx14] …
Private sums: A “straw-man” scheme

Assume that the servers are non-colluding. Equivalently: that at least one server is honest.
Private sums: A “straw-man” scheme

[Chaum88], [BGW88], …
[KDK11] [DFKZ13] [PrivEx14] …
Secret sharing
Pick three random “shares” that sum to 1.
\[ 1 = 15 + (-12) + (-2) \pmod{31} \]

Need all three shares to recover the shared value.
Private sums: A “straw-man” scheme

Secret sharing
Pick three random “shares” that sum to 1.

$$1 = 15 + (-12) + (-2) \quad \text{(mod 31)}$$

Need all three shares to recover the shared value.
Private sums:
A “straw-man” scheme

Secret sharing
Pick three random “shares” that sum to 1.
\[ 1 = 15 + (-12) + (-2) \pmod{31} \]

Need all three shares to recover the shared value.
Secret sharing
Pick three random “shares” that sum to 1.
\[ 1 = 15 + (-12) + (-2) \pmod{31} \]

Need all three shares to recover the shared value.
Private sums:
A “straw-man” scheme

Secret sharing
Pick three random “shares” that sum to 1.
\[ 1 = 15 + (-12) + (-2) \pmod{31} \]

Need all three shares to recover the shared value.
Private sums: A “straw-man” scheme
Private sums: A “straw-man” scheme
Private sums: A “straw-man” scheme

Server A: 15
Server B: -12
Server C: -2
Private sums: A “straw-man” scheme

\[ 0 = (-10) + 7 + 3 \]
Private sums: A “straw-man” scheme

\[ 0 = (-10) + 7 + 3 \]
Private sums: A “straw-man” scheme

\[ -10 + 7 + 3 = 0 = (-10) + 7 + 3 \]
Private sums: A “straw-man” scheme

Server A: 15
Server B: -12
Server C: -2

-10 7 3

0
Private sums: A “straw-man” scheme

Server A: 15
Server B: -12
Server C: -2

-10  7  3

0
Private sums: A “straw-man” scheme

Server A: 15-10
Server B: -12+7
Server C: -2+3

0
Private sums:
A “straw-man” scheme

Server A
15-10+…

Server B
-12+7+…

Server C
-2+3+…
Private sums:
A “straw-man” scheme
Private sums: A “straw-man” scheme

\[ S_A + S_B + S_C = 15 + (-10) + \ldots \]
Private sums:
A “straw-man” scheme

\[ S_A + S_B + S_C = 15 + (-10) + \ldots \]
Private sums: A “straw-man” scheme

\[ S_A + S_B + S_C = 15 + (-10) + \ldots \]
\[ = 1 + 0 + \ldots + 1 \]
Private sums:
A “straw-man” scheme

\[ S_A + S_B + S_C = 15 + (-10) + \ldots + 1 \]

Servers learn the sum of client values and learn *nothing else.*
Private sums:
A “straw-man” scheme

$S_A + S_B + S_C = 15 + (-10) + \ldots\nonumber$ 
$= 1 + 0 + \ldots + 1$

Servers learn the sum of client values and learn *nothing else.*
Private sums: A “straw-man” scheme

\[ S_A + S_B + S_C = 15 + (-10) + \ldots \]
\[ = 1 + 0 + \ldots + 1 \]

Learn that three phones are on the Bay Bridge—don’t know which three
Computing private sums
Computing private sums

**Exact correctness:** If everyone follows the protocol, servers compute the sum of all $x_i$s.

**Privacy:** Any proper subset of the servers learns nothing but the sum of the $x_i$s.

**Efficiency:** Follows by inspection.
Computing private sums

**Exact correctness:** If everyone follows the protocol, servers compute the sum of all $x_i$s.

**Privacy:** Any proper subset of the servers learns nothing but the sum of the $x_i$s.

**Efficiency:** Follows by inspection.

**Robustness:** ???
Private sums: A “straw-man” scheme

Server A: 15 - 10
Server B: -12 + 7
Server C: -2 + 3
Private sums: A “straw-man” scheme

15 - 10
-12 + 7
-2 + 3

x is supposed to be a 0/1 value
Private sums: A "straw-man" scheme
Private sums: A “straw-man” scheme

Server A: 15-10
Server B: -12+7
Server C: -2+3
Private sums:
A “straw-man” scheme

An evil client needn’t follow the rules!
Private sums: A “straw-man” scheme

An evil client needn’t follow the rules!

10 + 4 + 7 = 21
Private sums:
A “straw-man” scheme
Private sums: A “straw-man” scheme

Server A

Server B

Server C

garbage

garbage

garbage
Private sums: A “straw-man” scheme

A single bad client can undetectably corrupt the sum

Users have incentives to cheat

Typical defenses (NIZKs) are costly
Outline

• Background: The private aggregation problem
• A straw-man solution for private sums
• Providing robustness with SNIPs
• Evaluation
• Encodings for complex aggregates
Outline

• Background: The private aggregation problem

• A straw-man solution for private sums

• Providing robustness with SNIPs

• Evaluation

• Encodings for complex aggregates
Contribution 1
Secret-shared non-interactive proofs (SNIPs)

\[ x = 1 \]
Contribution 1
Secret-shared non-interactive proofs (SNIPs)

\[ x = 1 \]
Contribution 1
Secret-shared non-interactive proofs (SNIPs)

15 + (-12) + (-2) = 1

x = 1
Contribution 1
Secret-shared non-interactive proofs (SNIPs)

15  -12  -2
x = 1
Contribution 1
Secret-shared non-interactive proofs (SNIPs)

\[ x = 1 \]
Contribution 1
Secret-shared non-interactive proofs (SNIPs)

The servers want to ensure that their shares sum to 0 or 1
...without learning x.
Contribution 1
Secret-shared non-interactive proofs (SNIPs)

More generally, servers
• hold shares of the client’s private value $x$
• hold an arbitrary public predicate $Valid(\cdot)$ – expressed as an arithmetic circuit
• want to test if “$Valid(x)$” holds, without leaking $x$
Contribution 1
Secret-shared non-interactive proofs (SNIPs)

More generally, servers
- hold shares of the client’s private value \( x \)
- hold an arbitrary public predicate \( \text{Valid}(\cdot) \) – expressed as an arithmetic circuit
- want to test if \( \text{Valid}(x) \)

For our running example:
\( \text{Valid}(x) = “x \in \{0,1\}” \)
Contribution 1
Secret-shared non-interactive proofs (SNIPs)

More generally, servers
• hold shares of the client’s private value \( x \)
• hold an arbitrary public predicate \( \text{Valid}(\cdot) \)
  – expressed as an arithmetic circuit
• want to test if “Valid(\(x\))” holds, without leaking \( x \)
Contribution 1
Secret-shared non-interactive proofs (SNIPs)

\[ x = 1 \]
Contribution 1
Secret-shared non-interactive proofs (SNIPs)

\[ x = 1 \]
Contribution 1
Secret-shared non-interactive proofs (SNIPs)
Contribution 1
Secret-shared non-interactive proofs (SNIPs)
Contribution 1
Secret-shared non-interactive proofs (SNIPs)

\[ x = 1 \]
Contribution 1
Secret-shared non-interactive proofs (SNIPs)

\[ x = 1 \]
Contribution 1
Secret-shared non-interactive proofs (SNIPs)

Servers gossip

$x = 1$
Contribution 1
Secret-shared non-interactive proofs (SNIPs)

\[ x = 1 \]
Contribution 1
Secret-shared non-interactive proofs (SNIPs)

\[ \pi_a, x_a \quad \pi_b, x_b \quad \pi_c, x_c \]

\( x = 1 \)


**Contribution 1**

Secret-shared non-interactive proofs (SNIPs)

\[\pi_a, x_a, \pi_b, x_b, \pi_c, x_c\]

\[x = 1\]
Contribution 1
Secret-shared non-interactive proofs (SNIPs)

x = 1
Contribution 1
Secret-shared non-interactive proofs (SNIPs)

\[ x = 1 \]
Contribution 1
Secret-shared non-interactive proofs (SNIPs)

\[ x = 1 \]
Contribution 1
Secret-shared non-interactive proofs (SNIPs)

\[ x = 1 \]
**Contribution 1**

Secret-shared non-interactive proofs (SNIPs)

Fail

\[ \pi_a, X_a \quad \pi_b, X_b \quad \pi_c, X_c \]

\[ x = 1 \]
Contribution 1
Secret-shared non-interactive proofs (SNIPs)

Fail

$\Pi_a, X_a$

Fail

$\Pi_b, X_b$

$\Pi_c, X_c$

$x = 1$
Contribution 1
Secret-shared non-interactive proofs (SNIPs)

Server A
Server B
Server C

Fail
Fail
Fail

\[ \pi_a, x_a \]
\[ \pi_b, x_b \]
\[ \pi_c, x_c \]

0

x = 1
Contribution 1
Secret-shared non-interactive proofs (SNIPs)

\[ x = 1 \]
**Contribution 1**
Secret-shared non-interactive proofs (SNIPs)

- Prio servers *detect and reject malformed client submissions*
- In this example, each client can influence the aggregate statistic by +/- 1, at most
How SNIPs work

The servers want to ensure that

\[ \text{Valid}(x) = \text{Valid}(x_a + x_b + x_c) = 1 \]

...without learning \( x \).
How SNIPs work

Server A

$X_a$

Server B

$X_b$

Server C

$X_c$
How SNIPs work

Could run **secure multiparty computation** to check that $\text{Valid}(x) = 1$.

[GMW87], [BGW88]
How SNIPs work

Could run **secure multiparty computation** to check that \( \text{Valid}(x) = 1 \).

[GMW87], [BGW88]
How SNIPs work

Server A

Server B

Server C

$X_a$

$X_b$

$X_c$
How SNIPs work
How SNIPs work

**Idea:** Client generates the transcripts that servers *would* have observed in a multi-party computation.

See also [IKOS07]
How SNIPs work

Idea: Client generates the transcripts that servers *would* have observed in a multi-party computation.

See also [IKOS07]
How SNIPs work

Server A
Server B
Server C

X

X_a
X_b
X_c
How SNIPs work
How SNIPs work

Servers check that the transcripts are valid and consistent.
How SNIPs work

Servers check that the transcripts are valid and consistent.
How SNIPs work

Servers check that the transcripts are valid and consistent.

Checking a transcript is much easier than generating it!
How SNIPs work

Server A

\[ \Pi_a \quad x_a \]

Server B

\[ \Pi_b \quad x_b \]

Server C

\[ \Pi_c \quad x_c \]
How SNIPs work
How SNIPs work

Server A

Server B

Server C

$\Pi_a \quad X_a \quad D_a$

$\Pi_b \quad X_b \quad D_b$

$\Pi_c \quad X_c \quad D_c$

“Randomized digest” of the transcript
How SNIPs work

Server A  Server B  Server C

\( \Pi_a \) \( X_a \) \( \Pi_b \) \( X_b \) \( \Pi_c \) \( X_c \)

\( D_a \) \( D_b \) \( D_c \)
How SNIPs work

Server A

Server B

Server C

$D_a$

$D_b$

$D_c$
How SNIPs work

Server A

Server B

Server C

$D_a$

$D_b$

$D_c$
How SNIPs work

- If $x$ is valid, $D_a + D_b + D_c = 0$
- If $x$ is invalid, $D_a + D_b + D_c \neq 0$ with high probability

Servers run lightweight multi-party computation to check that $D_a + D_b + D_c = 0$

If so, servers accept $x$ is valid.
How SNIPs work

- If \( x \) is valid, \( D_a + D_b + D_c = 0 \)
- If \( x \) is invalid, \( D_a + D_b + D_c \neq 0 \) with high probability

Servers run lightweight multi-party computation to check that

\[ D_a + D_b + D_c = 0 \]

If so, servers accept \( x \) is valid.

[BFO12]
<table>
<thead>
<tr>
<th>M = # of multiplication gates in \text{Valid}(\cdot) circuit</th>
<th>Public-key ops.</th>
<th>Communication</th>
<th>Slow-down</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Client</td>
<td>Server</td>
<td>C-to-S</td>
</tr>
<tr>
<td>M = # of multiplication gates in ( \text{Valid}(\cdot) ) circuit</td>
<td>Public-key ops.</td>
<td>Communication</td>
<td>Slow-down</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td></td>
<td>Client</td>
<td>Server</td>
<td>C-to-S</td>
</tr>
<tr>
<td>Dishonest-maj. MPC</td>
<td>0</td>
<td>( \Theta(M) )</td>
<td>0</td>
</tr>
</tbody>
</table>

[CLOS02], [DPSZ12], ...
<table>
<thead>
<tr>
<th>M = # of multiplication gates in ( \text{Valid}(\cdot) ) circuit</th>
<th>Public-key ops.</th>
<th>Communication</th>
<th>Slow-down</th>
</tr>
</thead>
<tbody>
<tr>
<td>Client</td>
<td>Server</td>
<td>C-to-S</td>
<td>S-to-S</td>
</tr>
<tr>
<td>Dishonest-maj. MPC [CLOS02], [DPSZ12], …</td>
<td>(0)</td>
<td>(\Theta(M))</td>
<td>(0)</td>
</tr>
<tr>
<td>Commits + NIZKs [FS86], [CP92], [CS97], …</td>
<td>(\Theta(M))</td>
<td>(\Theta(M))</td>
<td>(\Theta(M))</td>
</tr>
<tr>
<td>M = # of multiplication gates in $\text{Valid}(\cdot)$ circuit</td>
<td>Public-key ops.</td>
<td>Communication</td>
<td>Slow-down</td>
</tr>
<tr>
<td>---------------------------------------------------------------</td>
<td>-----------------</td>
<td>---------------</td>
<td>-----------</td>
</tr>
<tr>
<td></td>
<td>Client</td>
<td>Server</td>
<td>C-to-S</td>
</tr>
<tr>
<td>Dishonest-maj. MPC</td>
<td>0</td>
<td>$\Theta(M)$</td>
<td>0</td>
</tr>
<tr>
<td>[CLOS02], [DPSZ12], ...</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Commits + NIZKs</td>
<td>$\Theta(M)$</td>
<td>$\Theta(M)$</td>
<td>$\Theta(M)$</td>
</tr>
<tr>
<td>[FS86], [CP92], [CS97], ...</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Commits + SNARKs</td>
<td>$\Theta(M)$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>[GGPR13], [BCGTV13], ...</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>M = # of multiplication gates in ( \text{Valid}(\cdot) ) circuit</td>
<td>Public-key ops.</td>
<td>Communication</td>
<td>Slow-down</td>
</tr>
<tr>
<td>-------------------------------------------------</td>
<td>----------------</td>
<td>--------------</td>
<td>-----------</td>
</tr>
<tr>
<td>Client</td>
<td>Server</td>
<td>C-to-S</td>
<td>S-to-S</td>
</tr>
<tr>
<td>Dishonest-maj. MPC</td>
<td>0</td>
<td>( \Theta(M) )</td>
<td>0</td>
</tr>
<tr>
<td>[CLOS02], [DPSZ12], …</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Commits + NIZKs</td>
<td>( \Theta(M) )</td>
<td>( \Theta(M) )</td>
<td>( \Theta(M) )</td>
</tr>
<tr>
<td>[FS86], [CP92], [CS97], …</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Commits + SNARKs</td>
<td>( \Theta(M) )</td>
<td>( \Theta(M) )</td>
<td>( \Theta(M) )</td>
</tr>
<tr>
<td>[GGPR13], [BCGTV13], …</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>This work: SNIPs</td>
<td>0</td>
<td>0</td>
<td>( \Theta(M) )</td>
</tr>
<tr>
<td>[CLOS02], [DPSZ12], …</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>M = # of multiplication gates in $\text{Valid}(\cdot)$ circuit</td>
<td>Public-key ops.</td>
<td>Communication</td>
<td>Slow-down</td>
</tr>
<tr>
<td>---------------------------------------------------------------</td>
<td>-----------------</td>
<td>---------------</td>
<td>-----------</td>
</tr>
<tr>
<td></td>
<td>Client</td>
<td>Server</td>
<td>C-to-S</td>
</tr>
<tr>
<td>Dishonest-maj. MPC</td>
<td>0</td>
<td>$\Theta(M)$</td>
<td>0</td>
</tr>
<tr>
<td>[CLOS02], [DPSZ12], ...</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Commits + NIZKs</td>
<td>$\Theta(M)$</td>
<td>$\Theta(M)$</td>
<td>$\Theta(M)$</td>
</tr>
<tr>
<td>[FS86], [CP92], [CS97], ...</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Commits + SNARKs</td>
<td>$\Theta(M)$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>[GGPR13], [BCGTV13], ...</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>This work: SNIPs</td>
<td>0</td>
<td>0</td>
<td>$\Theta(M)$</td>
</tr>
</tbody>
</table>

For specific $\text{Valid}()$ circuits, it is possible to eliminate this cost [BGI16]
<table>
<thead>
<tr>
<th>M = # of multiplication gates in ( \text{Valid}(\cdot) ) circuit</th>
<th>Public-key ops.</th>
<th>Communication</th>
<th>Slow-down</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Client</td>
<td>Server</td>
<td>C-to-S</td>
</tr>
<tr>
<td>Dishonest-maj. MPC</td>
<td>0</td>
<td>( \Theta(M) )</td>
<td>0</td>
</tr>
<tr>
<td>[CLOS02], [DPSZ12], …</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Commits + NIZKs</td>
<td>( \Theta(M) )</td>
<td>( \Theta(M) )</td>
<td>( \Theta(M) )</td>
</tr>
<tr>
<td>[FS86], [CP92], [CS97], …</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Commits + SNARKs</td>
<td>( \Theta(M) )</td>
<td>( O(1) )</td>
<td>( O(1) )</td>
</tr>
<tr>
<td>[GGPR13], [BCGTV13], …</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>This work: SNIPs</td>
<td>0</td>
<td>0</td>
<td>( \Theta(M) )</td>
</tr>
</tbody>
</table>
Outline

• Background: The private aggregation problem
• A straw-man solution for private sums
• **Providing robustness with SNIPs**
• Evaluation
• Encodings for complex aggregates
Outline

• Background: The private aggregation problem
• A straw-man solution for private sums
• Providing robustness with SNIPs
• Evaluation
• Encodings for complex aggregates
Evaluation

- Implemented Prio in Go
  (see optimizations described in paper)
- Five-server cluster in EC2
- System collects the sum of “N” 0/1 values

Four variants
1. No privacy
2. No robustness ("straw man")
3. Prio   (privacy + robustness)
4. NIZK   (privacy + robustness)
Evaluation

- Implemented Prio in Go
  (see optimizations described in paper)
- Five-server cluster in EC2
- System collects the sum
  of “N” 0/1 values

Four variants
1. No privacy
2. No robustness (“straw man”)
3. Prio (privacy + robustness)
4. NIZK (privacy + robustness)

E.g., for privately measuring telemetry data.
Evaluation

- Implemented Prio in Go  
  (see optimizations described in paper)
- Five-server cluster in EC2
- System collects the sum of “N” 0/1 values

Four variants
1. No privacy
2. No robustness (“straw man”)
3. Prio (privacy + robustness)
4. NIZK (privacy + robustness)
Evaluation

- Implemented Prio in Go (see optimizations described in paper)
- Five-server cluster in EC2
- System collects the sum of “N” 0/1 values

Four variants
1. No privacy
2. No robustness (“straw man”)
3. Prio (privacy + robustness)
4. NIZK (privacy + robustness)
Evaluation

• Implemented Prio in Go
  (see optimizations described in paper)

• Five-server cluster in EC2

• System collects the sum of “N” 0/1 values

Four variants
1. No privacy
2. No robustness (“straw man”)
3. Prio (privacy + robustness)
4. NIZK (privacy + robustness)
Five-server cluster in five Amazon data centers
Five-server cluster in five Amazon data centers

Submission length (0/1 integers)

Submissions processed/s

Prio

NIZK
Five-server cluster in five Amazon data centers

Submission length (0/1 integers)

Submissions processed/s

Prio

NIZK

Submission length (0/1 integers)
Five-server cluster in five Amazon data centers

Submission length (0/1 integers)

- 2^4
- 2^6
- 2^8
- 2^10
- 2^12
- 2^14
- 2^16

Submissions processed/s

- 10000
- 1000
- 100
- 10
- 1

Prio

NIZK

50x performance improvement
Five-server cluster in five Amazon data centers

Submission length (0/1 integers)

Submissions processed/s

- No robustness
- Prio
- NIZK
Submission length (0/1 integers)

Submissions processed/s

Five-server cluster in five Amazon data centers

No privacy

No robustness

Prio

NIZK
Five-server cluster in five Amazon data centers

Within 10x of no privacy

Submission length (0/1 integers)

Submissions processed/s

- NIZK
- Prio
- No privacy
- No robustness
Per-server data transfer

Submission length (0/1 integers)

2^2  2^6  2^{10}  2^{14}

256 B  4 KiB  64 KiB  1 MiB

NIZK
Submission length (0/1 integers)

Per-server data transfer

Servers exchange a constant number of bytes.
Outline

• Background: The private aggregation problem

• A straw-man solution for private sums

• Providing robustness with SNIPs

• Evaluation

• Encodings for complex aggregates
Outline

• Background: The private aggregation problem
• A straw-man solution for private sums
• Providing robustness with SNIPs
• Evaluation
• Encodings for complex aggregates
Known techniques: Complex statistics

If you can compute private sums, you can compute many other interesting aggregates using known techniques

- Average
- Variance
- Standard deviation
- Most popular (approx)
- “Heavy hitters” (approx)
- Min and max (approx)
- Quality of arbitrary regression model ($R^2$)
- Least-squares regression
- Stochastic gradient descent [Bonawitz et al. 2016]
Known techniques: Complex statistics

If you can compute private sums, you can compute many other interesting aggregates using known techniques

• Average
• Variance
• Standard deviation
• Most popular (approx)
• “Heavy hitters” (approx)
• Min and max (approx)
• Quality of arbitrary regression model ($R^2$)
• Least-squares regression
• Stochastic gradient descent  [Bonawitz et al. 2016]

Contribution 2: SNIP-friendly encodings for these statistics
Known techniques: Complex statistics

If you can compute private sums, you can compute many other interesting aggregates using known techniques

- Average
- Variance
- Standard deviation
- Most popular (approx)
- “Heavy hitters” (approx)
- Min and max (approx)
- Quality of arbitrary regression model ($R^2$)
- Least-squares regression
- Stochastic gradient descent [Bonawitz et al. 2016]

Contribution 2:
SNIP-friendly encodings for these statistics

Prio can’t compute all statistics efficiently
Known techniques: Complex statistics

If you can compute private sums, you can compute many other interesting aggregates using known techniques:

- Average
- Variance
- Standard deviation
- Most popular (approx)
- “Heavy hitters” (approx)
- Min and max (approx)
- Quality of arbitrary regression model ($R^2$)
- Least-squares regression
- Stochastic gradient descent

[PrivStats11], [KDK11], [DFKZ13], [PrivEx14], [MDD16], …

Contribution 2:
SNIP-friendly encodings for these statistics

Prio can’t compute all statistics efficiently

See the paper for the details

[Bonawitz et al. 2016]
Today

![StressTracker](image)

- Blood pressure
- Twitter usage

Graph showing the correlation between blood pressure and Twitter usage.
Today

[Graph showing the relationship between Blood pressure and Twitter usage]

StressTracker
With Prio…

Twitter usage

Blood pressure

App store

StressTracker
With Prio…

Blood pressure vs. Twitter usage chart.

App store

StressTracker
With Prio...

Blood pressure

Twitter usage

App store

StressTracker
With Prio…

Blood pressure

Twitter usage

\[ B(T) = c_1 \cdot T + c_0 \]
Conclusions

• Wholesale collection of sensitive user data puts our security at risk.

• Prio is the first system for aggregation that provides:
  – exact correctness,
  – privacy,
  – robustness, and
  – efficiency.

• To do so, Prio uses SNIPs and aggregatable encodings.

• These techniques together bring private aggregation closer to practical.

Thank you!

Henry Corrigan-Gibbs
henrycg@cs.stanford.edu

https://crypto.stanford.edu/prio/
Example Encoding: Average and Variance
Example Encoding: Average and Variance

- Each of N clients holds a value $x_i$
- Servers want the AVG and VAR of the $x_i$s.

Each client $i$ encodes her value $x$ as the pair $(x, y) = (x, x^2)$

Simple to check that the encoding is valid:
$$\text{Valid}(x, y) = (x^2 - y)$$  \[\text{outputs zero if valid}\]
Example Encoding: Average and Variance

- Each of N clients holds a value $x_i$
- Servers want the $AVG$ and $VAR$ of the $x_i$s.

Each client $i$ encodes her value $x$ as the pair 
$$(x, y) = (x, x^2)$$

Simple to check that the encoding is valid:
$$Valid(x, y) = (x^2 - y) \quad [\text{outputs zero if valid}]$$

Use Prio to compute the sum of encodings $\sum_i (x_i, y_i)$
Example Encoding: Average and Variance

- Each of N clients holds a value $x_i$
- Servers want the $AVG$ and $VAR$ of the $x_i$s.

Each client $i$ encodes her value $x$ as the pair

$$(x, y) = (x, x^2)$$

Simple to check that the encoding is valid:

$Valid(x, y) = (x^2 - y) \quad [\text{outputs zero if valid}]$

Use Prio to compute the sum of encodings $\sum_i (x_i, y_i)$

Then recover the statistics:

$$AVG(X) = \frac{\sum_i x_i}{N}$$

$$AVG(X^2) = \frac{\sum_i y_i}{N} = \frac{\sum_i x_i^2}{N}$$

$$VAR(X) = AVG(X^2) - AVG(X)^2$$
Client time (s)

Lower is better.

- **Heart**: 13 mixed features
- **BrCa**: 30x14-bit ints

<table>
<thead>
<tr>
<th></th>
<th>SNARK (Est.)</th>
<th>NIZK</th>
<th>Prio-MPC</th>
<th>Prio</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Heart</strong></td>
<td><strong>0.001</strong></td>
<td><strong>0.01</strong></td>
<td>0.1</td>
<td>0.1</td>
</tr>
<tr>
<td><strong>BrCa</strong></td>
<td><strong>0.0001</strong></td>
<td><strong>0.01</strong></td>
<td>0.01</td>
<td>0.01</td>
</tr>
</tbody>
</table>
Using 128-bit integers
Using 128-bit integers

Submit data
Using 128-bit integers

Submit data

$X_a, \Pi_a$
Using 128-bit integers

Submit data

\(x_a, \pi_a\)

\(x_b, \pi_b\)
Using 128-bit integers

Submit data

\[ x_a, \pi_a \]

\[ x_b, \pi_b \]

\[ x_c, \pi_c \]
Using 128-bit integers

Submit data

Proportional to length of data submission and size of “Valid” circuit

\[ x_a, \pi_a \] \rightarrow \[ x_b, \pi_b \] \rightarrow \[ x_c, \pi_c \]
Using 128-bit integers

Submit data

\( x_a, \pi_a \)

\( x_b, \pi_b \)

\( x_c, \pi_c \)
Using 128-bit integers
Using 128-bit integers

Submit data

$X_a, \Pi_a$

AES key

$X_b, \Pi_b$

AES key

$X_c, \Pi_c$
Using 128-bit integers

Check that $P(r) = 0$

Submit data

AES key

$X_a, \pi_a$

$X_b, \pi_b$

$X_c, \pi_c$

32 B

16 B

16 B

16 B

16 B

16 B

16 B
Using 128-bit integers

Submit data

Check that \( P(r) = 0 \)

Accept/reject client data

AES key

\( X_a, \pi_a \)

\( X_b, \pi_b \)

\( X_c, \pi_c \)

32 B

16 B

16 B

16 B
Using 128-bit integers

Submit data

Check that $P(r) \neq 0$

Accept/reject client data

AES key

AES key

$X_a, \Pi_a$

$X_b, \Pi_b$

$X_c, \Pi_c$

32 B

32 B

16 B

16 B

16 B

16 B

Ok/fail bit

Ok/fail bit
Submit data

Check that $P(r) = 0$

Accept/reject client data

Using 128-bit integers

Does not grow with size of data or “Valid” circuit
Using 128-bit integers

Submit data

Check that $P(r) = ? 0$

Accept/reject client data

$X_a, \Pi_a$

AES key

$X_b, \Pi_b$

AES key

$X_c, \Pi_c$

32 B

Ok/fail bit

32 B

16 B

Ok/fail bit

16 B

16 B

16 B

16 B
Example Encoding: Average and Variance
Example Encoding: Average and Variance

- Each of N clients holds a 4-bit value $x_i$
- Servers want the **AVG** and **VAR** of the $x_i$s.

Each client encodes her value $x = b_3 b_2 b_1 b_0$ as the tuple

$$(x, y) = (x, x^2, b_3, b_2, b_1, b_0)$$
Example Encoding: Average and Variance

- Each of N clients holds a 4-bit value $x_i$
- Servers want the AVG and VAR of the $x_i$s.

Each client encodes her value $x = b_3b_2b_1b_0$ as the tuple
$$(x, y) = (x, x^2, b_3, b_2, b_1, b_0)$$

To test validity of the encoding, check that:

$$\text{Valid}(x, y) = \begin{cases} 
(x^2 - y) = 0 & \text{--- } y \text{ is } x^2 \\
(x - \sum j2^i b_j) = 0 & \text{--- } b's \text{ are the bits of } x \\
b_j \cdot (b_j - 1) = 0 & \text{--- } b's \text{ are 0/1 values}
\end{cases}$$

[PrivStats11]