Riposte: An Anonymous Messaging System Handling Millions of Users

Paper by Henry Corrigan-Gibbs, Dan Boneh, David Mazieres

Slides by Kyle Soska
Communication Side Channels

- Authenticated encryption provides confidentiality, integrity, authenticity (CIA)

- Are there any other aspects of communication that could be exploited?
  - Timing information
  - Message sizes
  - Who is talking to who, packet endpoints
  - Active Behaviors such as responses to failures
Alice and Bob both sent some data at similar times in a low-latency anonymizing network so it could be either, however Charlie did not send anything so it could not have been him.
Timing Side Channels

• Problem: The time at which users choose to speak leaks information about their actions

• Idea: Mixnets try to mitigate this by queueing messages and re-ordering the outputs, tradeoff between latency and anonymity
Timing Side Channels

• Observation: Time is continuous, users can choose to send messages whenever they want
  – Continuous functions form distributions with high entropy, possibly lots of information

• Idea: Make time discrete. Break time into discrete fixed length epochs
Alice and Bob both sent data in the same epoch, while Charlie did not send anything, so Charlie is automatically known to not be the sender of the message.
Timing Side Channels

• Problem: Even if time is discrete, the fact that users don’t send information during each time epoch leaks information

• Idea: Each user that participates in the system is required to speak during each time epoch. If a user has nothing to say they still need to encrypt a “null message”
The amount of data of the received blue message disqualifies Alice and Charlie as the potential sender of the message.
Data Side Channels

• Problem: Sending large amounts of data within a single time epoch may leak information

• Idea: Each user should be required to send exactly the same amount of data during each epoch
High Level

• How can we make sure that there are no side channels that can leak information?
  – What is information?

• Will not try and answer that question instead we will try and make the observable distribution of actions from any two users in the system identical, then there are no side channels
  – Timing distribution fixed by epochs + required sending on each round
  – Sender size side channels fixed by requiring the same amount of data to be sent
  – Receiver size side channel fixed by PIR (from Riffle)
  – Etc.
Riposte is a protocol between a set of users and a set of servers that takes place during finite discrete time epochs.

Riposte Servers
# Servers = $s \geq 2$
# Malicious = $t < s$

Riposte Users
# Honest Users $\geq 2$
Riposte Step 1

- Users of the system who want to “tweet” a fixed-length message perform some interaction with the Riffle servers during a time epoch.

- These interactions will be computationally independent of the message that the user is tweeting.
Riposte Step 2

- The Riposte servers communicate and perform some processing on the data they received during the epoch.

- There are different ways to perform this step depending on the assumptions that the system is willing to make:
  - Efficient solutions exist if the servers trust a third party.
  - Slower solutions if no trusted third party exists.
Riposte Step 3

A public database is published by the Riposte servers which contain with high probability the complete list of messages that the users have “tweeted”.

- The president smells
- I am hungry
- Company X is committing crimes against humanity
Security Definition

- **Correctness**: The scheme is correct if, when all servers execute the protocol faithfully, the plaintext state of the database revealed at the end of the protocol run is equal to applying each valid client write request to an empty database.

- **Write Privacy**: An adversary’s advantage at guessing which honest client wrote some particular message in the database is negligibly better than guessing, even when the adversary controls $t$ of $s$ servers, and all but two clients.

- **Disruption Resistance**: We say that the protocol is disruption resistant if the probability that an adversary who makes $n$ write requests affects the state of more than $n$ rows in the database is negligible.
What is not Protected?

I am Alice, my key = K
My home address = Y
I was told if I tweet Z
I will win a free IPad

User behaviors that leak information are out of scope of the security definition for Riposte
High Level

• If we had a magical crypto wand that users could use to append their message to a public database in such a way that no adversary could associate any message with any user, would this be secure?

  – Correctness
  – Write Privacy
  – Disruption Resistance
Tools Used

• Private Information Writing (PIW)
  – Similar idea to Private Information Writing (PIR)

• Distributed Point Functions (DPFs)
  – Used to create efficient PIW schemes

• Pseudo Random Generators (PRGs)
  – Used to create DPFs for two servers

• Seed-Homomorphic PRGs
  – Used to create DPFs for more than two servers

• Secure Multi-party computation
  – Used to detect malformed write requests with a trusted third party
  – Won’t really do Secure MPC, but will draw inspiration from it

• Non-Interactive Zero Knowledge Proofs
  – Used to detect malformed write requests without a trusted third party

• Basic Probability
Private Information Writing

<table>
<thead>
<tr>
<th>Row</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
</tr>
</tbody>
</table>

$S_1$

$S_2$

<table>
<thead>
<tr>
<th>Row</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
</tr>
</tbody>
</table>
Private Information Writing

Suppose that each server has a pre-allocated database of fixed size $L$, initialized to 0s. The entries can take on values from $\mathbb{F}_2$, i.e. 0 or 1 and the user wants to write a ‘1’ into row $l$. 
User could simply tell one server the index of a row that he wishes to write a ‘1’ into.

This is very efficient, takes $O(\log(n))$ bits where $n$ is the # rows in the database.

This is NOT private, server $S_1$ knows the index that the user is setting to 1.
Private Information Writing

$r \leftarrow_R \mathbb{F}_2^5$

$r \oplus e_l$

<table>
<thead>
<tr>
<th>Row</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$l$</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Row</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$l$</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
</tr>
</tbody>
</table>
Private Information Writing

\[ S_1 \]

<table>
<thead>
<tr>
<th>Row</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>( l )</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
</tr>
</tbody>
</table>

\[ S_2 \]

<table>
<thead>
<tr>
<th>Row</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>( l )</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
</tr>
</tbody>
</table>

\[ r = \]

<table>
<thead>
<tr>
<th>Row</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>( x_0 \overset{R}{\leftarrow} \mathbb{F}_2 )</td>
</tr>
<tr>
<td>1</td>
<td>( x_1 \overset{R}{\leftarrow} \mathbb{F}_2 )</td>
</tr>
<tr>
<td>( l )</td>
<td>( x_l \overset{R}{\leftarrow} \mathbb{F}_2 )</td>
</tr>
<tr>
<td>3</td>
<td>( x_3 \overset{R}{\leftarrow} \mathbb{F}_2 )</td>
</tr>
<tr>
<td>4</td>
<td>( x_4 \overset{R}{\leftarrow} \mathbb{F}_2 )</td>
</tr>
</tbody>
</table>

\[ r = \]

<table>
<thead>
<tr>
<th>Row</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>( r[0] )</td>
</tr>
<tr>
<td>1</td>
<td>( r[1] )</td>
</tr>
<tr>
<td>( l )</td>
<td>( r[l] \oplus 1 )</td>
</tr>
<tr>
<td>3</td>
<td>( r[3] )</td>
</tr>
<tr>
<td>4</td>
<td>( r[4] )</td>
</tr>
</tbody>
</table>
### Private Information Writing

<table>
<thead>
<tr>
<th>Row</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>l</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Row</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>l</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Row</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$S_1[0] \oplus S_2[0] = 1 \oplus 1 = 0$</td>
</tr>
<tr>
<td>1</td>
<td>$S_1[1] \oplus S_2[1] = 0 \oplus 0 = 0$</td>
</tr>
<tr>
<td>2</td>
<td>$S_1[2] \oplus S_2[2] = 0 \oplus 1 = 1$</td>
</tr>
<tr>
<td>3</td>
<td>$S_1[3] \oplus S_2[3] = 1 \oplus 1 = 0$</td>
</tr>
<tr>
<td>4</td>
<td>$S_1[4] \oplus S_2[4] = 1 \oplus 1 = 0$</td>
</tr>
</tbody>
</table>
Private Information Writing

• From the point of view of a server, the input looks completely random, information theoretically secure.

• Problem: The communication complexity is $O(n)$ where $n$ is the size of the database.

• Solution: There are ways to express the “shares” using $O(\sqrt{n})$ space.

• Note: This approach generalizes to rows that take values other than $\mathbb{F}_2$ such as $\mathbb{F}_{2^{32}}$ (32-bit register) or $\mathbb{F}_{2^{10}}$ (approx. size of tweet).
Potential Problem

\begin{align*}
S_1 & \quad r \leftarrow_R \mathbb{F}_2^5 \quad r' \leftarrow_R \mathbb{F}_2^5 \\
 & \quad r \oplus e_l \quad r' \oplus e_l' \\
& \quad (r \leftarrow_R \mathbb{F}_2^5, r \oplus e_l) \quad (r' \leftarrow_R \mathbb{F}_2^5, r' \oplus e_l')
\end{align*}
Collisions

- Two different users could randomly pick the same entry in the database to write into, thus annihilating their contributions

- Idea: Make the database really large so this happens with an acceptable low probability

\[ \mathbb{E}[\%Success] \approx 1 - \frac{m}{n} + \frac{1}{2} \left(\frac{m}{2}\right)^2, \quad n \approx 19.5m \]  
where \( m \) is the number of users and \( n \) is the number of rows in the database

- \( m = 2^{10} = 1024 \) writers with 95% non-collisions requires \( n \approx 20,000 \) database rows
Collisions Optimization

- What if users could collide a little bit but not too much?

- Probability that $k$ users collide becomes small very rapidly as $k$ increases (super linear)
  - We would be willing to double the size of each row in the database if we could tolerate collisions of two users
Collisions Optimization

- Let each row in the database be in a field of odd characteristic, say \( \mathbb{F} = \mathbb{F}_p \) for \( p = 2^{64} - 59 \)

- When client \( A \) goes to write message \( m_A \in \mathbb{F} \), he writes in \( (m_A, m_A^2) \in \mathbb{F}^2 \)

- If no collision occurs then the final recovered value for this row is \( (m_A, m_A^2) \), it can be checked by squaring the first value and comparing it to the 2\(^{nd} \)

- If a collision occurs then the final recovered value is \( (m_A + m_B, m_A^2 + m_B^2) \)
  - Let \( S_1 = m_A + m_B \pmod{p} \), \( S_2 = m_A^2 + m_B^2 \pmod{p} \)
  - \( S_1^2 = m_A^2 - 2m_A m_B + m_B^2 \pmod{p} \), \( 2S_2 = 2m_A^2 + 2m_B^2 \pmod{p} \)
  - \( 2S_2 - S_1^2 = (m_A - m_B)^2 \pmod{p} \)
  - \( \therefore m_A - m_B = \sqrt{2S_2 - S_1^2} \), \( m_A = \frac{(S_1 + m_A - m_B)}{2} \), \( m_B = S_1 - m_A \)
Collision Optimization

- Tradeoff the length of each row in the database for collision resistance

- New probability with resistance against 2 collisions:

  \[ E[\%Success] \approx 1 - \frac{1}{2} \left( \frac{m}{n} \right)^2 + \frac{1}{3} \left( \frac{m}{n} \right)^3 \]

  - For 95% success need \( n \approx 2.7m \), for \( m = 2^{10} = 1024 \) writers, need \( n \approx 2,800 \) rows instead of \( n \approx 20,000 \) from before
Status

• Know how to do private information writing
  – Fixed row collisions

• What is left?
  – Private information writing is not efficient, each write request takes $O(n)$ space where $n$ is the size of the database

• Solution: We will fix this problem with a clever use of distributed point functions
Distributed Point Functions

- PIW Problem: The communication complexity is $O(n)$ where $n$ is the size of the database.

- Solution: There are ways to express the “shares” using $O(\sqrt{n})$ space.

- We will define a primitive called a distributed point function (DPF) and show that any distributed point function can be used to build a PIR scheme, and then show a DPF that uses $O(\sqrt{n})$ space.
Point Functions

• **Point Function**: Fix a positive integer $L$ and a finite field $\mathbb{F}$. For all $l \in \mathbb{Z}_L$ and $m \in \mathbb{F}$ the point function $P_{l,m} : \mathbb{Z}_L \to \mathbb{F}$ is the function such that $P_{l,m}(l) = m$ and $P_{l,m}(l') = 0$ for all $l \neq l'$

• **Point Function (Informal)**: For any row in a fixed-sized database and any valid message, the corresponding point function takes in a row number and outputs zero everywhere except the particular row where it outputs the message.

\[
P_{2,123} = \begin{bmatrix}
1 & 0 \\
2 & 123 \\
3 & 0 \\
4 & 0 \\
5 & 0
\end{bmatrix}
\]

\[
P_{5,1} = \begin{bmatrix}
1 & 0 \\
2 & 0 \\
3 & 0 \\
4 & 0 \\
5 & 1
\end{bmatrix}
\]
Distributed Point Functions

- **Distributed Point Function (DPF):** Fix a positive integer $L$ and a finite field $\mathbb{F}$. An $(s, t)$-distributed point function consists of a pair of possibly randomized algorithms that implement the following functionalities
  
  - $\text{Gen}(l, m) \rightarrow (k_0, \ldots, k_{s-1})$. Given an integer $l \in \mathbb{Z}_L$ and value $m \in \mathbb{F}$, output a list of $s$ keys
  - $\text{Eval}(k, l') \rightarrow m'$. Given a key $k$ generated using $\text{Gen}$ and an index $l' \in \mathbb{Z}_L$, return a value $m' \in \mathbb{F}$

- **Distributed Point Function (DPF) (Informal):** Given a message and a row in the database, an algorithm called $\text{Gen}$ will create a bunch of keys or “shares”, and each share can be evaluated at each row in the database
Distributed Point Functions

- **Correctness**: For a collection of $s$ keys generated using $\text{Gen}(l, m)$, the sum of the outputs of these keys using $\text{Eval}$ must be equal to the point function $P_{l,m}$
  \[
  \Pr[(k_0, \ldots, k_{s-1}) \leftarrow \text{Gen}(l, m): \sum_{i=0}^{s-1} \text{Eval}(k_i, l') = P_{l,m}(l')] = 1
  \]

- **Correctness (Informal)**: If you use Gen to generate a set of keys, and then you evaluate each key at some row and sum the results together, you will always get the same result you would have gotten from evaluating the non-distributed point function at that point.
Distributed Point Functions

- Privacy: Let $S$ be any subset of $\{0, \ldots, s-1\}$ such that $|S| \leq t$. Then for any $l \in \mathbb{Z}_L$ and $m \in \mathbb{F}$, let $D_{S,l,m}$ denote the distribution of keys $\{(k_i)|i \in S\}$ induced by $(k_0, \ldots, k_{s-1}) \leftarrow Gen(l, m)$. We say that an $(s, t) - DPF$ maintains privacy if there exists a p.p.t. algorithm $Sim$ such that the following distributions are computationally indistinguishable:
  - $D_{S,l,m} \approx_c Sim(S)$

- Privacy (Informal): For any subset of keys that an adversary might obtain, the adversary could simulate the distribution of keys themselves, meaning that they could not have possibly learned anything at all from observing this subset and so privacy is maintained
Using Distributed Point Functions

• Each user will generate a set of keys when they want to perform a write operation
  
  \[ \{k_0, ..., k_{s-1}\} \leftarrow Gen(l, m) \]

• The user will then take the keys and give one to each of the \( s \) servers

• Privacy tells us that no colluding set of servers can combine their shares together to learn anything about \( l \) or \( m \)

• Correctness tells us that when the servers combine their database states at the end of the epoch, the user’s message \( m \) will appear in row \( l \)
Private Information Writing With Distributed Point Functions

\[
\begin{array}{c|c}
\text{Row} & \text{Value} \\
0 & \text{Eval}(k_0, 0) \\
1 & \text{Eval}(k_0, 1) \\
2 & \text{Eval}(k_0, 2) \\
3 & \text{Eval}(k_0, 3) \\
4 & \text{Eval}(k_0, 4) \\
\end{array}
\]

\[
\begin{array}{c|c}
\text{Row} & \text{Value} \\
0 & \text{Eval}(k_1, 0) \\
1 & \text{Eval}(k_1, 1) \\
2 & \text{Eval}(k_1, 2) \\
3 & \text{Eval}(k_1, 3) \\
4 & \text{Eval}(k_1, 4) \\
\end{array}
\]

\[\{k_0, k_1\} \leftarrow \text{Gen}(l, m)\]
Distributed Point Functions

- Clearly if we can represent the keys compactly, then we have a way to do PIW efficiently.

- How can we represent the keys in a compact way, say $O(\sqrt{n})$?

- Idea: when we generated the keys in the example protocol in the beginning, the first key was $r \leftarrow_R \mathbb{F}_L^L$ where $L$ is the size of the database. This is a lot of randomness, we should find some way to compress all of this randomness.
  - Transmit the seed to a secure PRG instead and use the PRG to expand the seed to generate randomness.
Compact Distributed Point Functions

- Take the row indexes from the database and map them into a matrix
- Calculate two values, $l_x, l_y$ such that $l = l_x \cdot y + l_y$
  - $l = 2$: $l_x = 1$, $l_y = 0$
  - $l = 3$: $l_x = 1$, $l_y = 1$
Compact Distributed Point Functions

**Gen**(\(l, m\)) \(\rightarrow\) \((k_A, k_B)\):

- Let \(b_A \leftarrow_R \{0, 1\}^x = \{b_0, ..., b_{l_x}, ..., b_{x-1}\}\), length \(x\) vector of random bits
- Let \(s_A \leftarrow_R \mathbb{S}^x = \{s_0, ..., s_{l_x}, ..., s_{x-1}\}\), length \(x\) vector of random PRG seeds

- Let \(b_B = \{b_0, ..., \overline{b}_{l_x}, ..., b_{x-1}\}\), same as \(b_A\) but with position \(l_x\) negated
- Let \(s_B = \{s_0, ..., s_{l_x}^*, ..., s_{x-1}\}\), same as \(s_A\) but with position \(l_x\) swapped for \(s_{l_x}^*\) where \(s_{l_x}^* \leftarrow_R \mathbb{S}\)

- Let \(v \leftarrow m \ast e_{l_y} + G(s_{l_x}) + G(s_{l_x}^*)\), where \(G\) is a PRG that takes seeds and stretches them to length \(y\) vectors

Let \(l = 1\), then \(l_x = 0,\ l_y = 1\)

\[k_A = (b_A, s_A, v) = \begin{pmatrix} b_0 \\ s_0 \\ b_1 \\ s_1 \end{pmatrix} \quad k_B = (b_B, s_B, v) = \begin{pmatrix} \overline{b}_0 \\ s_0^* \\ b_1 \\ s_1 \end{pmatrix}\]
Compact Distributed Point Functions

- \( \text{Eval}(k, l') = \text{Eval}(b, s, v, l') \)
  - Calculate \( l'_x, l'_y \) from \( l' \)
  - Use \( G \) to stretch \( l'_x \)-th seed of \( s \) into a length \( y \) vector: \( g \leftarrow G(s[l'_x]) \)
  - Return \( m' \leftarrow (g[l'_y] + b[l'_x]v[l'_y]) \)
Compact Distributed Point Functions

- Adding together the evaluations of the two keys results in the desired write into an empty database
- Any row where \( b, s \) are the same are obviously going to be identical and so their values will be zero
- Any row where \( b, s \) are different, will be equal to \( G(s_i) + G(s_i^*) + v = 2G(s_i) + 2G(s_i^*) + m \cdot e_{ly} = m \cdot e_{ly} \)

\[
\begin{array}{c|c|c}
\hline
b_A & s_A & v \\
\hline
1 & s_0 & G(s_0) + v \\
0 & s_1 & G(s_1) \\
1 & s_2 & G(s_2) + v \\
1 & s_3 & G(s_3) + v \\
0 & s_4 & G(s_4) \\
0 & s_5 & G(s_5) \\
\end{array}
\]

\[
\begin{array}{c|c|c}
\hline
b_A & s_B & v \\
\hline
1 & s_0 & G(s_0) + v \\
0 & s_1 & G(s_1) \\
1 & s_2 & G(s_2) + v \\
0 & s^*_3 & G(s^*_3) \\
0 & s_4 & G(s_4) \\
0 & s_5 & G(s_5) \\
\end{array}
\]

\[
= 
\]

\[
\begin{array}{cccccccc}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & m \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{array}
\]
Compact Distributed Point Functions

• Do the keys reveal knowledge to the server about either the index or the message?
  – No because the first one is generated randomly, and the second one is like a one-time pad which is known to be random

• Are the keys compact?
  – \( k = (b_A, s_A, v) \)
  – \(|b_A| = x\)
  – \(|s_A| = x \cdot \alpha\) where \(\alpha = |seed|\)
  – \(|v| = y \cdot \beta\) where \(\beta = \log_2(p)\) when using \(\mathbb{F}_p\)
  – \(|k| = (1 + \alpha)x + \beta y = O(\sqrt{n}) + O(\sqrt{n}) = O(\sqrt{n})\)
Private Information Writing With Distributed Point Functions

\[ S_1 \]

\[
\begin{array}{|c|c|}
\hline
\text{Row} & \text{Value} \\
\hline
0 & \text{Eval}(k_0, 0) \\
1 & \text{Eval}(k_0, 1) \\
2 & \text{Eval}(k_0, 2) \\
3 & \text{Eval}(k_0, 3) \\
4 & \text{Eval}(k_0, 4) \\
\hline
\end{array}
\]

\[ S_2 \]

\[
\begin{array}{|c|c|}
\hline
\text{Row} & \text{Value} \\
\hline
0 & \text{Eval}(k_1, 0) \\
1 & \text{Eval}(k_1, 1) \\
2 & \text{Eval}(k_1, 2) \\
3 & \text{Eval}(k_1, 3) \\
4 & \text{Eval}(k_1, 4) \\
\hline
\end{array}
\]

\[ \{k_0, k_1\} \leftarrow \text{Gen}(l, m) \]
Status

• Know how to do private information writing
  – Fixed row collisions
  – Made it efficient for two servers where 1 of them may be malicious
    • Transmit the seeds of PRGs in a clever way to reduce the amount of randomness that needs to be sent and achieve $O(\sqrt{n})$ size

• What is left?
  – What if we wanted to use the “Anytrust” assumption where we have $s$ servers and $s - 1$ of them may be malicious?

• Solution:
  – The distributed point function that generates $s$ keys is going to look very similar to the distributed point function that generates 2 keys, with a few minor differences
Taste of Distributed Point Function for $s$ servers

- Generate $s - 1$ vectors of random bits $(b_0, \ldots, b_{s-2})$ and random seeds $(s_0, \ldots, s_{s-2})$

- Pick $b_{s-1}, s_{s-1}, v$ adaptively so that everything works out

- Evaluation works almost exactly like before except that correctness requires the PRG $G$ to have a very special property, namely that it is seed homomorphic
Seed Homomorphic PRG

- A seed-homomorphic PRG is a pseudo-random generator $G$ mapping seeds in a group $(\mathbb{S}, \oplus)$ to outputs in a group $(\mathbb{G}, \otimes)$ with the additional property that for any $s_0, s_1 \in \mathbb{S}$: $G(s_0 \oplus s_1) = G(s_0) \otimes G(s_1)$

- Example $\mathbb{G} = \mathbb{Z}_{13}$:

<table>
<thead>
<tr>
<th>Seed</th>
<th>$y_0$</th>
<th>$y_1$</th>
<th>$y_2$</th>
<th>$y_3$</th>
<th>$y_4$</th>
<th>$y_5$</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Seed</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$y_0$</th>
<th>$y_1$</th>
<th>$y_2$</th>
<th>$y_3$</th>
<th>$y_4$</th>
<th>$y_5$</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>7</td>
<td>1</td>
<td>11</td>
<td>8</td>
<td>10</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>8</td>
<td>9</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>9</td>
<td>9</td>
<td>7</td>
<td>12</td>
<td>11</td>
</tr>
</tbody>
</table>
Seed Homomorphic PRG

- Let $\mathbb{G}$ be a $q$-order elliptic curve subgroup
- Pick $y$ random points $(P_0, \ldots, P_{y-1}) \in \mathbb{G}^y$
- Take in a seed which will be a scalar $s \in \mathbb{Z}_q$

<table>
<thead>
<tr>
<th>Seed</th>
<th>$P_0$</th>
<th>$P_1$</th>
<th>$P_2$</th>
<th>$P_3$</th>
<th>$P_4$</th>
<th>$P_5$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$P_0$</td>
<td>$P_1$</td>
<td>$P_2$</td>
<td>$P_3$</td>
<td>$P_4$</td>
<td>$P_5$</td>
</tr>
<tr>
<td>2</td>
<td>$2P_0$</td>
<td>$2P_1$</td>
<td>$2P_2$</td>
<td>$2P_3$</td>
<td>$2P_4$</td>
<td>$2P_5$</td>
</tr>
<tr>
<td>3</td>
<td>$(2 + 1)P_0$</td>
<td>$(2 + 1)P_1$</td>
<td>$(2 + 1)P_2$</td>
<td>$(2 + 1)P_3$</td>
<td>$(2 + 1)P_4$</td>
<td>$(2 + 1)P_5$</td>
</tr>
</tbody>
</table>

- Regular PRGs can be instantiated with ciphers / symmetric crypto primitives, so quite fast.

- Seed-Homomorphic PRGs are only known to be constructed with public key primitives such as discrete logs, elliptic curves, or lattices, so orders of magnitude slower
Status

• Know how to do private information writing
  – Fixed row collisions
    • Make the database large enough that the probability of collisions is small
  – Made it efficient for two servers where 1 of them may be malicious
    • Transmit the seeds of PRGs in a clever way to reduce the amount of randomness that needs to be sent and achieve $O(\sqrt{n})$ size
  – Made it efficient for “anytrust” assumption with $s$ servers
    • Similar to before but now need slow homomorphic PRGs for correctness

• What is left?
  – Adversarial users could send fake keys that destroy the state of the server databases, effectively perform efficient DDOS

• Solution 1:
  – The servers use a trusted third party to make sure that the keys from any user are “well formed”

• Solution 2:
  – The user generates a non-interactive zero knowledge proof that the keys are well formed
Private Information Writing With Distributed Point Functions

If an adversary submitted a malformed write request, such as random shares for example, it would completely destroy the state of the database causing denial of service (DOS).
Secure Multi-Party Computation

- Any computation that can be performed between Alice and Bob with the assistance of a trusted third party can be computed directly between Alice and Bob.

I have $1,000  
I have $2,000

How can Alice and Bob figure out who has more money?

More generally, Alice and Bob want to compute some function on their secret inputs in such a way that nothing other than the output of the function is revealed.
Secure Multi-Party Computation

I have $1,000, b = 1,000

I have $2,000, a = 2,000

(a > b )?
Secure Multi-Party Computation

Alice has more money
\[ a > b = \text{True} \]

Bob

Alice

Alice has more money
\[ a > b = \text{True} \]
Secure Multi-Party Computation

Alice

Let's compute this circuit that takes our secret inputs

Bob

(a > b)

Okay

“Garbled Circuit” + Alice's secret “Garbled Input”

Interactive Coin-Flipping protocol to evaluate circuit

Answer: Alice has more money
Secure Multi-Party Computation

- If there is a trusted third party then we can evaluate a function on secret inputs fast

- If there is no trusted third party and we need to use garbled circuits and coin-flipping then things become very slow

- Assume that there exists a trusted third party, that will compute the function “AlmostEqual” using the secret shares or keys as inputs
AlmostEqual

- Servers $A$, $B$ want to know if the write request that they received is “well-formed” without revealing anything about the request other than if it is well-formed or not

- $k_A = (b_A, s_A, v)$, vector of random bits, vector of random seeds, fixed vector $v$

- $A$ forms a new vector $t_A$ where $t_A[i] = b_A[i] || s_A[i]$, just concatenate the seed and random bit at each row
AlmostEqual

- $B$ forms $t_B$ in exactly the same way, and the third party responds with “1” if $t_A, t_B$ differ in exactly one position, “0” otherwise

- Still need to know if $v$ is well formed
  - Not enough to simply check that $v$ is equal

- $A$ forms $u_A = \sum_{i=0}^{x-1} G(s_A[i])$
- $B$ forms $u_B = v + \sum_{i=0}^{x-1} G(s_B[i])$
- Run AlmostEqual($u_A, u_B$), “1” if valid, “0” if invalid

- Check: $u_B = v + \sum_{i=0}^{x-1} G(s_B[i]) = m \cdot e_{l_y} + G(s_A[l_x]) + G(s_B[l_x]) + \sum_{i=0}^{x-1} G(s_B[i])$
- $= m \cdot e_{l_y} + G(s_A[l_x]) + G(s_B[l_x]) + G(s_B[0]) + G(s_B[1]) + ... + G(s_B[x-1])$
- But $G(s_A[l'_x]) = G(s_B[l_x]) \forall l'_x \neq l_x$
- So $u_B = m \cdot e_{l_y} + \sum_{i=0}^{x-1} G(s_A[i]) = u_A + m \cdot e_{l_y}$ which differs from $u_A$ at exactly one position, namely position $l_y$
Detecting Malicious Writes

• The trusted third party has successfully detected malicious write requests

• What if we don’t have a trusted third party?
  – We can do cryptographic secure multi-party computation, but that is extremely inefficient
  – The users could submit zero-knowledge proofs along with their write requests to convince the servers that their requests are well-formed
Taste of the Zero Knowledge Proof

• Use Pedersen commitments over discrete logs
  – Similar to Homework 5 problem 1
  – Computationally Binding
  – Perfectly Hiding

• Let $g, h$ be generators of the standard prime order quadratic residue subgroup modulo a strong prime, $p = 2q + 1$, $p, q$ prime, $|QR(\mathbb{Z}_p)| = q$

• User commits to a message $m$ by forming $g^m h^r \pmod{p}$ where $r \leftarrow \mathbb{Z}_q$

• Notice these commitments are homomorphic:
  $g^{m_1} h^{r_1} \cdot g^{m_2} h^{r_2} = g^{m_1 + m_2} h^{r_1 + r_2}$

• Do some discrete log tricks with this commitment scheme, exploiting the homomorphic property to convince the servers that the write request is well-formed
Status

- Know how to do private information writing
  - Fixed row collisions
    - Make the database large to drive down the probability of random collisions
  - Made it efficient for two servers where 1 of them may be malicious
    - Transmit the seeds of PRGs in a clever way to reduce the amount of randomness that needs to be sent and achieve $O(\sqrt{n})$ size
  - Made it efficient for “anytrust” assumption with $s$ servers
    - Similar to before but now need slow homomorphic PRGs for correctness
  - Detected Malicious Write Requests
    - With TTP: use efficient computation with third party
    - Without TTP: use expensive NI-ZKP generated by users
    - Without TTP: could do cryptographic MPC but its slower than the ZK proofs

- We are done everything has been accounted for, what remains is to evaluate the performance
Fig. 3: As the database table size grows, the throughput of our system approaches the maximum possible given the AES throughput of our servers.
Evaluation

Fig. 5: The total client and server data transfer scales sub-linearly with the size of the database.
Evaluation

Fig. 6: Throughput of an eight-server Riposte cluster using the (8, 7)-distributed point function.