Private Set Intersection

Mike Rosulek

crypt@b-it 2018
Private set intersection (PSI)

Special case of secure 2-party computation:

\[ X = \{x_1, x_2, \ldots\} \]

\[ Y = \{y_1, y_2, \ldots\} \]

\[ X \cap Y \]
PSI applications

Contact discovery, when signing up for WhatsApp
  ▶ $X = \text{address book in my phone (phone numbers)}$
  ▶ $Y = \text{WhatsApp user database}$
PSI applications

Contact discovery, when signing up for WhatsApp
- $X =$ address book in my phone (phone numbers)
- $Y =$ WhatsApp user database

Private scheduling
- $X =$ available timeslots on my calendar
- $Y =$ available timeslots on your calendar
PSI applications

Contact discovery, when signing up for WhatsApp
   ▶ \( X = \) address book in my phone (phone numbers)
   ▶ \( Y = \) WhatsApp user database

Private scheduling
   ▶ \( X = \) available timeslots on my calendar
   ▶ \( Y = \) available timeslots on your calendar

Ad conversion rate (PSI variant)
   ▶ \( X = \) users who saw the advertisement
   ▶ \( Y = \) customers who bought the product
PSI applications

Contact discovery, when signing up for WhatsApp
- $X = \text{address book in my phone (phone numbers)}$
- $Y = \text{WhatsApp user database}$

Private scheduling
- $X = \text{available timeslots on my calendar}$
- $Y = \text{available timeslots on your calendar}$

Ad conversion rate (PSI variant)
- $X = \text{users who saw the advertisement}$
- $Y = \text{customers who bought the product}$

No-fly list
- $X = \text{passenger list of flight 123}$
- $Y = \text{government no-fly list}$
“Obvious” protocol

\[ x_1, x_2, \ldots \]

\[ H(x_1), H(x_2), \ldots \]

\[ y_1, y_2, \ldots \]

compare: \( H(y_1), \ldots \)

I wonder if she had item insecure:

Receiver can test \( \frac{x_1, \ldots, x_n}{x_1, \ldots, x_n} \), offline

Problematic if items have low entropy (e.g., phone numbers).
“Obvious” protocol

\[ x_1, x_2, \ldots \]

\[ H(x_1), H(x_2), \ldots \]

\[ y_1, y_2, \ldots \]

I wonder if she had item \(v\)

**INSECURE:** Receiver can test *any* \(v \in \{x_1, \ldots, x_n\}\), offline

- Problematic if items have low entropy (e.g., phone numbers)
Classical protocol [Meadows86, HubermanFranklinHogg99]

special case: each party has just one item
Classical protocol \([Meadows86, HubermanFranklinHogg99]\)

special case: each party has **just one** item

\[
\begin{align*}
x & \xrightarrow{H(x)^{\alpha}} y \\
H(y)^{\beta}; H(x)^{\alpha \beta} & \xleftarrow{} H(x)^{\alpha \beta} \quad \text{check: } H(x)^{\alpha \beta} = H(y)^{\beta \alpha}
\end{align*}
\]

Idea:

- If \(x = y\), then \(H(x)^{\alpha \beta} = H(y)^{\beta \alpha}\)
- If \(x \neq y\), they are independently random (when \(H\) is random oracle)
Classical protocol  

\[ \text{[Meadows86, HubermanFranklinHogg99]} \]

\[ \begin{align*}
& x_1, x_2, \ldots \quad H(x_1)^\alpha, H(x_2)^\alpha, \ldots \\
& \quad H(y_1)^\beta, H(x_2)^\beta, \ldots \\
& \quad H(x_1)^\alpha \beta, H(x_2)^\alpha \beta, \ldots
\end{align*} \]


\[ y_1, y_2, \ldots \]

Idea:

- If \( x = y \), then \( H(x)^\alpha \beta = H(y)^\beta \alpha \)
- If \( x \neq y \), they are independently random (when \( H \) is random oracle)
Classical protocol \cite{Meadows86,HubermanFranklinHogg99}

\[ x_1, x_2, \ldots \]

\[ H(x_1)^\alpha, H(x_2)^\alpha, \ldots \]

\[ H(y_1)^\beta, H(x_2)^\beta, \ldots \]

\[ H(x_1)^\alpha^\beta, H(x_2)^\alpha^\beta, \ldots \]

\[ y_1, y_2, \ldots \]

Idea:
- If \( x = y \), then \( H(x)^{\alpha\beta} = H(y)^{\beta\alpha} \)
- If \( x \neq y \), they are independently random (when \( H \) is random oracle)

Drawback: \( O(n) \) expensive exponentiations
Roadmap

1. **Crypto**: Private equality tests:
   - How to securely test whether two strings are identical
   - Focus on building from OT (and similar primitives) in light of OT extension

2. **Algorithmic**: Hashing techniques
   - How to reduce number of equality tests
Simplest case: string equality

\[ x \quad \text{does } x = y? \quad y \]
Simplest case: string equality

Using Yao’s protocol: \((x, y \in \{0, 1\}^\ell)\)

- \(\ell\) OTs
- Boolean circuit with \(\ell - 1\) AND gates
- E.g.: \(\ell = 64 \Rightarrow 48\) Kbits
String equality from OT

\[
\begin{array}{cc}
m_{1,0} & m_{1,1} \\
m_{2,0} & m_{2,1} \\
m_{3,0} & m_{3,1} \\
m_{4,0} & m_{4,1} \\
\vdots & \vdots \\
\end{array}
\]

- Sender chooses \(2\ell\) random strings

\[x = 0101, \quad m_{1,0} ; 0 \quad ? \quad m_{2,1} ; 1 \quad ? \quad m_{3,1} ; 1 \quad ? \quad m_{4,0} ; 0 \quad \vdots \]

\[y = 0110\]
String equality from OT

- Sender chooses $2\ell$ random strings
- Receiver uses bits of $y$ as OT choice bits
String equality from OT

\[ x = 0101 \cdots \]
\[
\begin{array}{c|c}
 m_{1,0} & m_{1,1} \\
 m_{2,0} & m_{2,1} \\
 m_{3,0} & m_{3,1} \\
 m_{4,0} & m_{4,1} \\
 \vdots & \vdots \\
\end{array}
\]

\[ y = 0110 \cdots \]
\[
\begin{array}{c|c}
 m_{1,0} & ? \\
 ? & m_{2,1} \\
 ? & m_{3,1} \\
 m_{4,0} & ? \\
 \vdots & \vdots \\
\end{array}
\]

- Sender chooses \(2\ell\) random strings
- Receiver uses bits of \(y\) as OT choice bits
- **Summary value** of \(v\) defined as \(\bigoplus_i m_{i,v_i}\)
  - Sender can compute **any** summary value (in particular, for \(x\))
  - Receiver can compute summary value **only** for \(y\)
  - Summary values other than \(y\) look random to receiver

\[ m_{1,0} \oplus m_{2,1} \oplus m_{3,0} \oplus \cdots \]
String equality from OT

\[ x = 0101 \ldots \]

\[
\begin{array}{cc}
m_{1,0} & m_{1,1} \\
m_{2,0} & m_{2,1} \\
m_{3,0} & m_{3,1} \\
m_{4,0} & m_{4,1} \\
\vdots & \vdots \\
\end{array}
\]

\[ y = 0110 \ldots \]

\[
\begin{array}{cc}
m_{1,0} & ? \\
? & m_{2,1} \\
? & m_{3,1} \\
m_{4,0} & ? \\
\vdots & \vdots \\
\end{array}
\]

- Sender chooses \( 2\ell \) random strings
- Receiver uses bits of \( y \) as OT choice bits
- **Summary value** of \( v \) defined as \( \bigoplus_i m_i v_i \)
  - Sender can compute **any** summary value (in particular, for \( x \))
  - Receiver can compute summary value only for \( y \)
  - Summary values other than \( y \) look random to receiver

**Cost:** just \( \ell \) OTs
Improving equality tests [PinkasSchneiderZohner14]

**Idea:** Instead of binary inputs, use base-\( k \) (base 3 in this example)

- Now only \( \log_k \ell \) instances of 1-out-of-\( k \) OT
Improving equality tests

$$x = 2101 \cdots$$

<table>
<thead>
<tr>
<th>$m_{1,0}$</th>
<th>$m_{1,1}$</th>
<th>$m_{1,2}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$m_{2,0}$</td>
<td>$m_{2,1}$</td>
<td>$m_{2,2}$</td>
</tr>
<tr>
<td>$m_{3,0}$</td>
<td>$m_{3,1}$</td>
<td>$m_{3,2}$</td>
</tr>
<tr>
<td>$m_{4,0}$</td>
<td>$m_{4,1}$</td>
<td>$m_{4,2}$</td>
</tr>
<tr>
<td>$\vdots$</td>
<td>$\vdots$</td>
<td>$\vdots$</td>
</tr>
</tbody>
</table>

$$y = 0122 \cdots$$

<table>
<thead>
<tr>
<th>$m_{1,0}$</th>
<th>$?$</th>
<th>$?$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$?$</td>
<td>$m_{2,1}$</td>
<td>$?$</td>
</tr>
<tr>
<td>$?$</td>
<td>$?$</td>
<td>$m_{3,2}$</td>
</tr>
<tr>
<td>$?$</td>
<td>$?$</td>
<td>$m_{4,2}$</td>
</tr>
<tr>
<td>$\vdots$</td>
<td>$\vdots$</td>
<td>$\vdots$</td>
</tr>
</tbody>
</table>

**Idea:** Instead of binary inputs, use base-$k$ (base 3 in this example)

- Now only $\log_k \ell$ instances of 1-out-of-$k$ OT
- **Note:** Only random OT required
Improving equality tests [PinkasSchneiderZohner14]

\[ x = 2101 \ldots \]

\[
\begin{array}{ccc}
    m_{1,0} & m_{1,1} & m_{1,2} \\
    m_{2,0} & m_{2,1} & m_{2,2} \\
    m_{3,0} & m_{3,1} & m_{3,2} \\
    m_{4,0} & m_{4,1} & m_{4,2} \\
    \vdots & \vdots & \vdots
\end{array}
\]

\[ y = 0122 \ldots \]

\[
\begin{array}{ccc}
    m_{1,0} & ? & ? \\
    ? & m_{2,1} & ? \\
    ? & ? & m_{3,2} \\
    ? & ? & m_{4,2} \\
    \vdots & \vdots & \vdots
\end{array}
\]

Idea: Instead of binary inputs, use base-\( k \) (base 3 in this example)

- Now only \( \log_k \ell \) instances of 1-out-of-\( k \) OT
- Note: Only random OT required

Costs for different 1-out-of-\( k \) random OTs:

- Basic OT extension: \( k = 2 \):
  \( 128 \) bits/OT
- [KolesnikovKumaresan13]: \( k = 2^8 \Rightarrow \) 3\( \times \) fewer OTs \@ 256 bits/OT
- [OrruOrsiniScholl16]: \( k = 2^{76} \Rightarrow \) 76\( \times \) fewer OTs \@ 512 bits/OT
- [KolesnikovKumaresanRosulekTrieu16]: \( k = \infty \Rightarrow \) \(~ 480 \) bits total
Another generalization

\[
\begin{align*}
x &= 0101 \cdots \\
n_1,0 &\quad n_1,1 \\
n_2,0 &\quad n_2,1 \\
n_3,0 &\quad n_3,1 \\
n_4,0 &\quad n_4,1 \\
\vdots &\quad \vdots
\end{align*}
\]

Private equality test: Alice has \( x \), Bob has \( y \), Bob learns \( x \equiv y \)
Another generalization

Private equality test: Alice has $x$, Bob has $y$, Bob learns $x = y$

Private set membership: Alice has set $X$, Bob has $y$, Bob learns $y \in X$
1

**Crypto:** Private equality tests:
- How to securely test whether *two strings* are identical

2

**Algorithmic:** Hashing techniques
- How to reduce number of equality tests
Building block

\[ S = \{s_1, \ldots\} \]

\[ \text{Cost: } 1 \text{ OT primitive } + \text{ sending } n \text{ summary values} \]
Dumb solution

\[ X = \{ x_1, \ldots \} \]

\[ Y = \{ y_1, \ldots \} \]

\[ \text{Cost: } O(n^2) \]
Dumb solution

\[ X = \{x_1, \ldots\} \]

\[ Y = \{y_1, \ldots\} \]

\[ \text{Cost: } O(n^2) \]
Better approach w/ hashing

Agree on a random hash function $h : \{0, 1\}^* \rightarrow [m]$
Better approach w/ hashing

Agree on a random hash function \( h : \{0, 1\}^* \rightarrow [m] \)

Assign item \( v \) to bin \( \# h(v) \)

\[
h(x_1) = 7
\]

Cost:

\[
\sum_i O(a_i b_i)
\]

where \( a_i ; b_i = \) number of items in bin \( \# i \)

Except, this is completely insecure!
Better approach w/ hashing

Agree on a random hash function \( h : \{0, 1\}^* \rightarrow [m] \)

Assign item \( v \) to bin \# \( h(v) \)

\[
h(x_2) = 6
\]

Cost:
\[
\sum_i O(a_i b_i)
\]

Where \( a_i, b_i \) = number of items in bin \# \( i \)

Except, this is completely insecure! (why?)
Better approach w/ hashing

Agree on a random hash function \( h : \{0, 1\}^* \rightarrow [m] \)

Assign item \( v \) to bin \# \( h(v) \)

\[
\begin{array}{ccccccc}
& & & x_6 & & & \\
& & & & x_3 & & \\
& & & x_2, x_4 & & & \\
& & & x_1 & & & \\
& & x_5 & & & & \\
\end{array}
\]

Cost: \( \sum_i O(a_i b_i) \) where \( a_i, b_i \) = number of items in bin \# \( i \)

Except, this is completely insecure! (why?)
Better approach w/ hashing

Agree on a random hash function $h : \{0, 1\}^* \rightarrow [m]$

Assign item $v$ to bin # $h(v)$

<table>
<thead>
<tr>
<th>$x_6$</th>
<th>$y_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_3$</td>
<td>$y_1, y_6$</td>
</tr>
<tr>
<td>$x_2, x_4$</td>
<td>$y_3, y_5$</td>
</tr>
<tr>
<td>$x_1$</td>
<td></td>
</tr>
<tr>
<td>$x_5$</td>
<td>$y_2$</td>
</tr>
</tbody>
</table>
Better approach w/ hashing

Agree on a random hash function \( h : \{0, 1\}^* \rightarrow [m] \)

Assign item \( v \) to bin \( # h(v) \)

Do \( \Theta(n^2) \) PSI in each bin

**Idea:** if both parties share an item \( v \), both will put it in bin \( h(v) \)
Better approach w/ hashing

Agree on a random hash function $h : \{0, 1\}^* \rightarrow [m]$.

Assign item $v$ to bin # $h(v)$.

Do $\Theta(n^2)$ PSI in each bin.

Idea: if both parties share an item $v$, both will put it in bin $h(v)$.

Cost: $\sum_i O(a_i b_i)$ where $a_i, b_i =$ number of items in bin $#i$.

- With $n$ items into $n$ bins, $E[\text{cost}] = O(n)$!
Better approach w/ hashing

Agree on a random hash function $h : \{0, 1\}^* \rightarrow [m]$

Assign item $v$ to bin $\# h(v)$

Do $\Theta(n^2)$ PSI in each bin

Idea: if both parties share an item $v$, both will put it in bin $h(v)$

Cost: $\sum_i O(a_i b_i)$ where $a_i, b_i =$ number of items in bin $\#i$

$\triangleright$ With $n$ items into $n$ bins, $E[\text{cost}] = O(n)$!

Except, this is completely insecure! (why?)
Subtleties with hashing

“cost = \( \sum_i O(a_i b_i) \)” ??

- **only if** \( a_i, b_i \) **public**
Subtleties with hashing

“cost = $\sum_i O(a_i b_i)$” ??

- only if $a_i, b_i$ public
Subtleties with hashing

“cost = \( \sum_i O(a_ib_i) \)” ??

- only if \( a_i, b_i \) public

1 item

1 item

2 items

1 item

1 item

she has no \( x \) with \( h(x) = 3 \)
Subtleties with hashing

“cost = \[ \sum_i O(a_ib_i) \]” ??

- only if \( a_i, b_i \) public

Solution:

1. Compute \( B \) such that \( \Pr_h[\text{no bin has } > B \text{ items}] \leq 2^{-s} \) (balls in bins)
2. Add **dummy items** so that each bin has **exactly** \( B \) items

\( \Rightarrow \) # (apparent) items per bin does not depend on input.
- (Protocol fails with probability \( 2^{-s} \))
Balls & bins questions

randomly assign

\( n \) balls \( \sim \) \( m \) bins

- Expected # balls per bin is \( n/m \)
- What is the **worst case** # balls in a bin (with high probability)?
Balls & bins questions

\[ \text{randomly assign} \]

\[ n \text{ balls} \sim m \text{ bins} \]

- Expected # balls per bin is \( n/m \)
- What is the **worst case** # balls in a bin (with high probability)?

**Natural parameter choice:** \( n \) items, \( n \) bins

- **Expected** balls per bin = 1
- **Worst-case** balls per bin = \( O(\log n) \)
- PSI cost = (# bins) \( \times \) (worst-case load)\(^2\) = \( O(n\log^2 n) \)
Balls & bins questions

\[ \text{n balls} \quad \sim \quad \text{m bins} \]

- Expected # balls per bin is \( n/m \)
- What is the **worst case** # balls in a bin (with high probability)?

**Natural parameter choice**: \( n \) items, \( n \) bins

- **Expected** balls per bin = 1
- **Worst-case** balls per bin = \( O(\log n) \)
- PSI cost = (# bins) \( \times \) (worst-case load)\(^2\) = \( O(n \log^2 n) \)

**Better parameter choice**: \( n \) items, \( O(n/\log n) \) bins  

- **Expected** balls per bin = \( O(\log n) \)
- **Worst-case** balls per bin = \( O(\log n) \)
- PSI cost = \( O(n \log n) \)
Improved hashing

Remember:

$$S = \{ s_1, \ldots \}$$

Our basic building block naturally supports one item from Bob
Improved hashing

Remember:

\[ S = \{s_1, \ldots\} \]

Our basic building block naturally supports **one item** from Bob

**Idea:** find hashing scheme that leaves only **1 item per bin**

- Only Bob needs to have 1 item per bin
Cuckoo hashing

Use 2 random hash functions $h_1, h_2$
Cuckoo hashing

Use 2 random hash functions $h_1, h_2$

If either $h_1(y)$ or $h_2(y)$ is empty,
   ▶ put $y$ in that bin
Cuckoo hashing

Use 2 random hash functions $h_1$, $h_2$

If either $h_1(y)$ or $h_2(y)$ is empty,
  ▶ put $y$ in that bin
Cuckoo hashing

Use 2 random hash functions $h_1, h_2$

If either $h_1(y)$ or $h_2(y)$ is empty,
  ▶ put $y$ in that bin

Claim: with sufficient bins, this process terminates with high probability.
Cuckoo hashing

Use 2 random hash functions $h_1, h_2$

If either $h_1(y)$ or $h_2(y)$ is empty,
  ▶ put $y$ in that bin

Claim: with sufficient bins, this process terminates with high probability.
Cuckoo hashing

Use 2 random hash functions $h_1, h_2$

If either $h_1(y)$ or $h_2(y)$ is empty,
  ▶ put $y$ in that bin

If $h_1(y)$ and $h_2(y)$ both occupied,
  ▶ **evict** someone $y'$ and recurse on $y'$
Cuckoo hashing

Use 2 random hash functions $h_1, h_2$

If either $h_1(y)$ or $h_2(y)$ is empty,
  ▶ put $y$ in that bin

If $h_1(y)$ and $h_2(y)$ both occupied,
  ▶ evict someone $y'$ and recurse on $y'$
Cuckoo hashing

Use 2 random hash functions $h_1, h_2$

If either $h_1(y)$ or $h_2(y)$ is empty,
  ▶ put $y$ in that bin

If $h_1(y)$ and $h_2(y)$ both occupied,
  ▶ evict someone $y'$ and recurse on $y'$

Claim: with sufficient bins, this process terminates with high probability.
Cuckoo hashing

Use 2 random hash functions $h_1, h_2$

If either $h_1(y)$ or $h_2(y)$ is empty,
  ▶ put $y$ in that bin

If $h_1(y)$ and $h_2(y)$ both occupied,
  ▶ evict someone $y'$ and recurse on $y'$
**Cuckoo hashing**

Use 2 random hash functions $h_1, h_2$

If either $h_1(y)$ or $h_2(y)$ is empty,
- put $y$ in that bin

If $h_1(y)$ and $h_2(y)$ both occupied,
- **evict** someone $y'$ and recurse on $y'$

**Claim:** with sufficient bins, this process terminates with high probability
Cuckoo hashing for PSI [PinkasSchneiderZohner14]

Agree on $h_1, h_2$

Bob hashes with Cuckoo hashing

Idea: Only Bob gets output from PMT. He places $y$ in $h(y)$; if Alice also has $y$, it will also be here.

Important: Alice cannot learn whether Bob placed $y$ in $h_1(y)$ or $h_2(y)$. 
Cuckoo hashing for PSI

Agree on $h_1, h_2$

Bob hashes with Cuckoo hashing

What about Alice?

Idea: Only Bob gets output from PMT

He places $y$ in $h(y)$; if Alice also has $y$, it will also be here

Important: Alice cannot learn whether Bob placed $y$ in $h_1(y)$ or $h_2(y)$.
Cuckoo hashing for PSI [PinkasSchneiderZohner14]

Agree on $h_1, h_2$

Bob hashes with Cuckoo hashing

What about Alice?

- Place $x$ in both $h_1(x)$ and $h_2(x)$
Cuckoo hashing for PSI [PinkasSchneiderZohner14]

Agree on $h_1, h_2$

Bob hashes with Cuckoo hashing

What about Alice?
- Place $x$ in both $h_1(x)$ and $h_2(x)$

| $x_6, x_1$ | $y_4$ |
| $x_6$ | $y_6$ |
| $x_1, x_3$ | $y_1$ |
| $x_3, x_4$ | $y_3$ |
| $x_2, x_4$ |   |
| $x_5$ | $y_5$ |
| $x_5, x_2$ | $y_2$ |
Cuckoo hashing for PSI [PinkasSchneiderZohner14]

Agree on $h_1, h_2$

Bob hashes with Cuckoo hashing

What about Alice?
  ▶ Place $x$ in both $h_1(x)$ and $h_2(x)$

PMT in each bin

<table>
<thead>
<tr>
<th></th>
<th>→ PMT →</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_6, x_1$</td>
<td>$y_4$</td>
</tr>
<tr>
<td>$x_6$</td>
<td>$y_6$</td>
</tr>
<tr>
<td>$x_1, x_3$</td>
<td>$y_1$</td>
</tr>
<tr>
<td>$x_3, x_4$</td>
<td>$y_3$</td>
</tr>
<tr>
<td>$x_2, x_4$</td>
<td></td>
</tr>
<tr>
<td>$x_5$</td>
<td>$y_5$</td>
</tr>
<tr>
<td>$x_5, x_2$</td>
<td>$y_2$</td>
</tr>
</tbody>
</table>
Cuckoo hashing for PSI

[PinkasSchneiderZohner14]

Agree on $h_1, h_2$

Bob hashes with Cuckoo hashing

What about Alice?

▶ Place $x$ in both $h_1(x)$ and $h_2(x)$

PMT in each bin

Idea: Only Bob gets output from PMT

▶ He places $y$ in $h_? (y)$; if Alice also has $y$, it will also be here

Important: Alice cannot learn whether Bob placed $y$ in $h_1(y)$ or $h_2(y)$
Cuckoo hashing PSI details

**Don’t forget:** Alice should pad with dummy items! (2n balls in m bins)

<table>
<thead>
<tr>
<th></th>
<th>← PMT →</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_6, x_1$</td>
<td></td>
<td>$y_4$</td>
</tr>
<tr>
<td>$x_6$</td>
<td></td>
<td>$y_6$</td>
</tr>
<tr>
<td>$x_1, x_3$</td>
<td></td>
<td>$y_1$</td>
</tr>
<tr>
<td>$x_3, x_4$</td>
<td></td>
<td>$y_3$</td>
</tr>
<tr>
<td>$x_2, x_4$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$x_5$</td>
<td></td>
<td>$y_5$</td>
</tr>
<tr>
<td>$x_5, x_2$</td>
<td></td>
<td>$y_2$</td>
</tr>
</tbody>
</table>
Don’t forget: Alice should pad with dummy items! (2n balls in m bins)
Cuckoo hashing PSI details

Don’t forget: Alice should pad with dummy items! (2n balls in m bins)
- Bob too!

| \( \bot, \bot \) | PMT | \( \bot' \) |
| \( x_6, x_1 \) | PMT | \( y_4 \) |
| \( x_6, \bot \) | PMT | \( y_6 \) |
| \( x_1, x_3 \) | PMT | \( y_1 \) |
| \( x_3, x_4 \) | PMT | \( y_3 \) |
| \( x_2, x_4 \) | PMT | \( \bot' \) |
| \( x_5, \bot \) | PMT | \( y_5 \) |
| \( x_5, x_2 \) | PMT | \( y_2 \) |
Cuckoo hashing PSI details

Don’t forget: Alice should pad with dummy items! (2n balls in m bins)

- Bob too!

Cost:

- ~ 1.5n bins for Cuckoo
- At most $O(\log n)$ items per bin for Alice

⇒ Still $O(n \log n)$ cost!
Avoiding dummy items

Summary values can be sent all together no longer associated with bins.

Previously:
- Can't leak true items in a bin

Now:
- Everyone knows: $n_{\text{true}}$ items
- Send only summary masks of true items
- Send only summary values, shuffled

Symbols:
- $x_1, x_2, x_3, x_4, x_5, \perp, \perp'$
- $y_1, y_2, y_3, y_4, y_5, y_6$
Avoiding dummy items

Summary values can be sent all together no longer associated with bins.

Previously:
- Can't leak \( n \) true items in a bin

Now:
- Everyone knows: \( n \) true items
  
  Send only summary masks of true items

\[ \begin{align*}
\bot, \bot & \xrightarrow{\text{ROT}} \bot' \\
x_6, x_1 & \xrightarrow{\text{ROT}} y_4 \\
x_6, \bot & \xrightarrow{\text{ROT}} y_6 \\
x_1, x_3 & \xrightarrow{\text{ROT}} y_1 \\
x_3, x_4 & \xrightarrow{\text{ROT}} y_3 \\
x_2, x_4 & \xrightarrow{\text{ROT}} \bot' \\
x_5, \bot & \xrightarrow{\text{ROT}} y_5 \\
x_5, x_2 & \xrightarrow{\text{ROT}} y_2
\end{align*} \]
Avoiding dummy items

Summary values can be sent all together

\[ x_6, x_1 \]
\[ x_6, \perp \]
\[ x_1, x_3 \]
\[ x_3, x_4 \]
\[ x_2, x_4 \]
\[ x_5, \perp \]
\[ x_5, x_2 \]

\[ \perp, \perp \]
\[ \perp' \]
\[ y_4 \]
\[ y_6 \]
\[ y_1 \]
\[ y_3 \]
\[ y_5 \]
\[ y_2 \]

all summary values, shuffled
Avoiding dummy items

Summary values can be sent all together

- No longer associated with bins

Previously:
- Can’t leak # true items in a bin

Now:
- Everyone knows: \( n \text{ true} \) items \( \Rightarrow 2n \text{ true} \) summary masks
- \( \Rightarrow \) Send only summary masks of true items

all summary values, shuffled
Cuckoo PSI costs

Other details:
- Actually use Cuckoo hashing with 3 hash functions

Costs:
- ~ 1.5n Cuckoo bins
- ~ 1.5n OT primitives
- 2n summary masks
⇒ total cost $O(n)$

Performance: [KolesnikovKumaresanRosulekTrieu16] = most efficient 1-out-of-$\infty$ OT equality test
- PSI of 1 million items ⇒ 3.8 seconds @ 120 MB
- Insecure protocol (hash and send) ⇒ 0.7 seconds @ 10 MB