**what is private set intersection (PSI)?**

<table>
<thead>
<tr>
<th>Alice</th>
<th>Bob</th>
</tr>
</thead>
<tbody>
<tr>
<td>e u r</td>
<td>c o v</td>
</tr>
<tr>
<td>o c r</td>
<td>i d l</td>
</tr>
<tr>
<td>y p t</td>
<td>g s u</td>
</tr>
<tr>
<td>x</td>
<td>x</td>
</tr>
</tbody>
</table>
what is private set intersection (PSI)?

Alice

![Diagram of private set intersection (PSI)]

Bob
what is private set intersection (PSI)?
what is private set intersection (PSI)?

Alice

?? u ??
co ?? ??
?? ?? ??

Bob

c o v
i d 1
9 s u
x

what is private set intersection (PSI)?

\{\text{my phone contacts}\} \cap \{\text{users of your service}\}

The Difficulty Of Private Contact Discovery

moxie0 on 03 Jan 2014

Building a social network is not easy. Social networks have value proportional to their size, so participants aren't motivated to join new social networks which aren't already large. It's a paradox where if people haven't already joined, people aren't motivated to join.
what is private set intersection (PSI)?

{my passwords} \cap \{passwords found in breaches\}
what is private set intersection (PSI)?

$\{\text{people who saw ad}\} \cap \{\text{customers who made purchases}\}$
state of the art: PSI for 1 million items:

Running time (s) vs communication (MB)

- Classic DH
- Semi-honest
- Malicious

Insecure hashing:

KKRT16 = ia.cr/2016/799
RR17a = ia.cr/2016/746
RR17b = ia.cr/2017/769
PRTY19 = ia.cr/2019/634
DKT10 = ia.cr/2010/469
state of the art: PSI for 1 million items:

- DKT10 = ia.cr/2010/469
- KKRT16 = ia.cr/2016/799
- RR17a = ia.cr/2016/746
- RR17b = ia.cr/2017/769
- PRTY19 = ia.cr/2019/634

Running time vs. communication for different protocols:
- Semi-honest vs. malicious: ~3x faster, 8x less communication.
- Semi-honest vs. semi-honest: 25% slower, 60–150% more communication.

Asymptotically, malicious PSI is infeasible due to the cost of OT extension.
state of the art: PSI for 1 million items:

vs prior malicious:
- \( \sim 3 \times \) faster
- \( 8 \times \) less comm

vs semi-honest:
- 25\% slower
- 60–150\% more comm

asymptotically:
- first \( O(n) \) malicious from OT extension

DKT10 = ia.cr/2010/469
KKRT16 = ia.cr/2016/799
RR17a = ia.cr/2016/746
RR17b = ia.cr/2017/769
PRTY19 = ia.cr/2019/634
1. why is existing semi-honest PSI so efficient?
2. why is malicious security harder?
3. how do we overcome this limitation?
1. why is existing semi-honest PSI so efficient?

2. why is malicious security harder?

3. how do we overcome this limitation?

what does “PaXoS” mean?
**batch oblivious PRF (OPRF)**

<table>
<thead>
<tr>
<th>Alice</th>
<th>Bob</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>

F learns nothing about x_i's all other F_i(x_i) look random achieved very efficiently from OT extension
**batch oblivious PRF (OPRF)**

<table>
<thead>
<tr>
<th>Alice</th>
<th>Bob</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_1$</td>
<td>1</td>
</tr>
<tr>
<td>$x_2$</td>
<td>2</td>
</tr>
<tr>
<td>$x_3$</td>
<td>3</td>
</tr>
<tr>
<td>$x_4$</td>
<td>4</td>
</tr>
<tr>
<td>$x_5$</td>
<td>5</td>
</tr>
<tr>
<td>$x_6$</td>
<td>6</td>
</tr>
<tr>
<td>$x_7$</td>
<td>7</td>
</tr>
<tr>
<td>$x_8$</td>
<td>8</td>
</tr>
<tr>
<td>$x_9$</td>
<td>9</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

 learns nothing about $x_i$'s all other $F_i(x^*)$ look random achieved very efficiently from OT extension
**batch oblivious PRF (OPRF)**

<table>
<thead>
<tr>
<th>Alice</th>
<th>Bob</th>
</tr>
</thead>
<tbody>
<tr>
<td>$F_1(x_1)$</td>
<td>$F_1(\cdot)$</td>
</tr>
<tr>
<td>$F_2(x_2)$</td>
<td>$F_2(\cdot)$</td>
</tr>
<tr>
<td>$F_3(x_3)$</td>
<td>$F_3(\cdot)$</td>
</tr>
<tr>
<td>$F_4(x_4)$</td>
<td>$F_4(\cdot)$</td>
</tr>
<tr>
<td>$F_5(x_5)$</td>
<td>$F_5(\cdot)$</td>
</tr>
<tr>
<td>$F_6(x_6)$</td>
<td>$F_6(\cdot)$</td>
</tr>
<tr>
<td>$F_7(x_7)$</td>
<td>$F_7(\cdot)$</td>
</tr>
<tr>
<td>$F_8(x_8)$</td>
<td>$F_8(\cdot)$</td>
</tr>
<tr>
<td>$F_9(x_9)$</td>
<td>$F_9(\cdot)$</td>
</tr>
</tbody>
</table>

Learns nothing about $x_i$'s all other $F_i(x_\ast)$ look random achieved very efficiently from OT extension
**batch oblivious PRF (OPRF)**

<table>
<thead>
<tr>
<th>Alice</th>
<th>Bob</th>
</tr>
</thead>
<tbody>
<tr>
<td>$F_1(x_1)$</td>
<td>1 $F_1(\cdot)$</td>
</tr>
<tr>
<td>$F_2(x_2)$</td>
<td>2 $F_2(\cdot)$</td>
</tr>
<tr>
<td>$F_3(x_3)$</td>
<td>3 $F_3(\cdot)$</td>
</tr>
<tr>
<td>$F_4(x_4)$</td>
<td>4 $F_4(\cdot)$</td>
</tr>
<tr>
<td>$F_5(x_5)$</td>
<td>5 $F_5(\cdot)$</td>
</tr>
<tr>
<td>$F_6(x_6)$</td>
<td>6 $F_6(\cdot)$</td>
</tr>
<tr>
<td>$F_7(x_7)$</td>
<td>7 $F_7(\cdot)$</td>
</tr>
<tr>
<td>$F_8(x_8)$</td>
<td>8 $F_8(\cdot)$</td>
</tr>
<tr>
<td>$F_9(x_9)$</td>
<td>9 $F_9(\cdot)$</td>
</tr>
</tbody>
</table>

... learns nothing about $x_i$’s
**batch oblivious PRF (OPRF)**

<table>
<thead>
<tr>
<th>Alice</th>
<th>Bob</th>
</tr>
</thead>
<tbody>
<tr>
<td>$F_1(x_1)$</td>
<td>$F_1(\cdot)$</td>
</tr>
<tr>
<td>$F_2(x_2)$</td>
<td>$F_2(\cdot)$</td>
</tr>
<tr>
<td>$F_3(x_3)$</td>
<td>$F_3(\cdot)$</td>
</tr>
<tr>
<td>$F_4(x_4)$</td>
<td>$F_4(\cdot)$</td>
</tr>
<tr>
<td>$F_5(x_5)$</td>
<td>$F_5(\cdot)$</td>
</tr>
<tr>
<td>$F_6(x_6)$</td>
<td>$F_6(\cdot)$</td>
</tr>
<tr>
<td>$F_7(x_7)$</td>
<td>$F_7(\cdot)$</td>
</tr>
<tr>
<td>$F_8(x_8)$</td>
<td>$F_8(\cdot)$</td>
</tr>
<tr>
<td>$F_9(x_9)$</td>
<td>$F_9(\cdot)$</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

All other $F_i(x^*)$ look random.

Bob learns nothing about $x_i$'s.
**batch oblivious PRF (OPRF)**

<table>
<thead>
<tr>
<th>Alice</th>
<th>Bob</th>
</tr>
</thead>
<tbody>
<tr>
<td>$F_1(x_1)$</td>
<td>$F_1(\cdot)$</td>
</tr>
<tr>
<td>$F_2(x_2)$</td>
<td>$F_2(\cdot)$</td>
</tr>
<tr>
<td>$F_3(x_3)$</td>
<td>$F_3(\cdot)$</td>
</tr>
<tr>
<td>$F_4(x_4)$</td>
<td>$F_4(\cdot)$</td>
</tr>
<tr>
<td>$F_5(x_5)$</td>
<td>$F_5(\cdot)$</td>
</tr>
<tr>
<td>$F_6(x_6)$</td>
<td>$F_6(\cdot)$</td>
</tr>
<tr>
<td>$F_7(x_7)$</td>
<td>$F_7(\cdot)$</td>
</tr>
<tr>
<td>$F_8(x_8)$</td>
<td>$F_8(\cdot)$</td>
</tr>
<tr>
<td>$F_9(x_9)$</td>
<td>$F_9(\cdot)$</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

All other $F_i(x^*)$ look random.

learns nothing about $x_i$'s.

Achieved very efficiently from OT extension.
the KKRT16 (PSZ14) protocol

<table>
<thead>
<tr>
<th>Alice</th>
<th>Bob</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>c</td>
</tr>
<tr>
<td>b</td>
<td>d</td>
</tr>
<tr>
<td>c</td>
<td>e</td>
</tr>
<tr>
<td>d</td>
<td>f</td>
</tr>
</tbody>
</table>
the KKRT16 (PSZ14) protocol

<table>
<thead>
<tr>
<th>Alice</th>
<th>m bins</th>
<th>Bob</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>1. Agree on random</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$h_1, h_2 : {0, 1}^* \rightarrow [m]$</td>
</tr>
<tr>
<td>a</td>
<td>1</td>
<td>c</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>d</td>
</tr>
<tr>
<td>b</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td>e</td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>c</td>
<td>8</td>
<td>f</td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
the KKRT16 (PSZ14) protocol

1. Agree on random \( h_1, h_2: \{0, 1\}^* \to [m] \)

2. Alice places each \( x \) into bin \( h_1(x) \) or \( h_2(x) \)

3. Bob places each \( x \) into bins \( h_1(x) \) and \( h_2(x) \)

4. OPRF in each bin:
   - Alice learns one \( F_i(x) \)
   - Bob learns entire \( F_i(\cdot) \)

5. Bob sends all \( F_i(x) \) values
the KKRT16 (PSZ14) protocol

1. Agree on random $h_1, h_2 : \{0, 1\}^* \rightarrow [m]$

<table>
<thead>
<tr>
<th>Alice</th>
<th>$m$ bins</th>
<th>Bob</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>1</td>
<td>c</td>
</tr>
<tr>
<td>b</td>
<td>2</td>
<td>d</td>
</tr>
<tr>
<td>c</td>
<td>3</td>
<td>e</td>
</tr>
<tr>
<td>d</td>
<td>4</td>
<td>f</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td></td>
</tr>
<tr>
<td></td>
<td>6</td>
<td></td>
</tr>
<tr>
<td></td>
<td>7</td>
<td></td>
</tr>
<tr>
<td></td>
<td>8</td>
<td></td>
</tr>
<tr>
<td></td>
<td>9</td>
<td></td>
</tr>
<tr>
<td></td>
<td>10</td>
<td></td>
</tr>
</tbody>
</table>
the KKRT16 (PSZ14) protocol

1. Agree on random $h_1, h_2 : \{0, 1\}^* \rightarrow [m]$
the KKRT16 (PSZ14) protocol

1. Agree on random $h_1, h_2 : \{0, 1\}^* \rightarrow [m]$

2. Alice places each $x$ into bin $h_1(x)$ or $h_2(x)$
the KKRT16 (PSZ14) protocol

1. Agree on random $h_1, h_2 : \{0, 1\}^* \rightarrow [m]$

2. Alice places each $x$ into bin $h_1(x)$ or $h_2(x)$

3. Bob places each $x$ into bins $h_1(x)$ and $h_2(x)$
**the KKRT16 (PSZ14) protocol**

1. **Alice** and **Bob** agree on random
   \[ h_1, h_2 : \{0, 1\}^* \rightarrow [m] \]

2. Alice places each \( x \) into bin \( h_1(x) \) or \( h_2(x) \)

3. Bob places each \( x \) into bins \( h_1(x) \) and \( h_2(x) \)

4. OPRF in each bin:
   - Alice learns one \( F_i(x) \);
   - Bob learns entire \( F_i(\cdot) \)
the KKRT16 (PSZ14) protocol

1. Agree on random $h_1, h_2 : \{0, 1\}^* \rightarrow [m]$

2. Alice places each $x$ into bin $h_1(x)$ or $h_2(x)$

3. Bob places each $x$ into bins $h_1(x)$ and $h_2(x)$

4. OPRF in each bin:
   - Alice learns one $F_i(x)$
   - Bob learns entire $F_i(\cdot)$

5. Bob sends all $F_i(x)$ values
the KKRT16 (PSZ14) protocol

Alice

1. Agree on random \( h_1, h_2 : \{0, 1\}^* \rightarrow [m] \)

2. Alice places each \( x \) into bin \( h_1(x) \) or \( h_2(x) \)

3. Bob places each \( x \) into bins \( h_1(x) \) and \( h_2(x) \)

4. OPRF in each bin:
   - Alice learns one \( F_i(x) \);
   - Bob learns entire \( F_i(\cdot) \)

5. Bob sends all \( F_i(x) \) values

Bob

\( m \) bins

1. \( F_1(\cdot) \)
2. \( F_2(\cdot) \)
3. \( F_3(\cdot) \)
4. \( F_4(\cdot) \)
5. \( F_5(\cdot) \)
6. \( F_6(\cdot) \)
7. \( F_7(\cdot) \)
8. \( F_8(\cdot) \)
9. \( F_9(\cdot) \)
10. \( F_{10}(\cdot) \)

\( \{ F_3(c), F_3(e) \} \)
the KKRT16 (PSZ14) protocol

<table>
<thead>
<tr>
<th>Alice</th>
<th>m bins</th>
<th>Bob</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>(F_1(\cdot))</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>(F_2(\cdot))</td>
</tr>
<tr>
<td></td>
<td>3, 4</td>
<td>(F_3(\cdot)), (F_4(\cdot))</td>
</tr>
<tr>
<td></td>
<td>5, 6</td>
<td>(F_5(\cdot)), (F_6(\cdot))</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>(F_7(\cdot))</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>(F_8(\cdot))</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>(F_9(\cdot))</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>(F_{10}(\cdot))</td>
</tr>
</tbody>
</table>

1. Agree on random \(h_1, h_2 : \{0, 1\}^* \rightarrow [m]\)

2. Alice places each \(x\) into bin \(h_1(x)\) or \(h_2(x)\)

3. Bob places each \(x\) into bins \(h_1(x)\) and \(h_2(x)\)

4. OPRF in each bin:
   Alice learns one \(F_i(x)\);
   Bob learns entire \(F_i(\cdot)\)

5. Bob sends all \(F_i(x)\) values

\(\{F_3(c), F_3(e), F_4(d)\}\)
the KKRT16 (PSZ14) protocol

1. Alice and Bob agree on random \( h_1, h_2 : \{0, 1\}^* \rightarrow [m] \)

2. Alice places each \( x \) into bin \( h_1(x) \) or \( h_2(x) \)

3. Bob places each \( x \) into bins \( h_1(x) \) and \( h_2(x) \)

4. OPRF in each bin:
   - Alice learns one \( F_i(x) \);
   - Bob learns entire \( F_i(\cdot) \)

5. Bob sends all \( F_i(x) \) values
the KKRT16 (PSZ14) protocol

1. Agree on random $h_1, h_2 : \{0, 1\}^* \rightarrow [m]$

2. Alice places each $x$ into bin $h_1(x)$ or $h_2(x)$

3. Bob places each $x$ into bins $h_1(x)$ and $h_2(x)$

4. OPRF in each bin:
   - Alice learns one $F_i(x)$;
   - Bob learns entire $F_i(\cdot)$

5. Bob sends all $F_i(x)$ values
why isn’t it secure against malicious parties?
why isn’t it secure against *malicious* parties?

<table>
<thead>
<tr>
<th>Alice</th>
<th>Bob</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$F_1(·)$</td>
</tr>
<tr>
<td>2</td>
<td>$F_2(·)$</td>
</tr>
<tr>
<td>3</td>
<td>$F_3(·)$</td>
</tr>
<tr>
<td>4</td>
<td>$F_4(·)$</td>
</tr>
<tr>
<td>5</td>
<td>$F_5(·)$</td>
</tr>
<tr>
<td>6</td>
<td>$F_6(·)$</td>
</tr>
<tr>
<td>7</td>
<td>$F_7(·)$</td>
</tr>
<tr>
<td>8</td>
<td>$F_8(·)$</td>
</tr>
<tr>
<td>9</td>
<td>$F_9(·)$</td>
</tr>
<tr>
<td>10</td>
<td>$F_{10}(·)$</td>
</tr>
</tbody>
</table>

\[
\{ F_3(c), F_3(e), F_4(d), \ldots, F_7(c), \ldots \} 
\]
why isn’t it secure against malicious parties?

Alice

Bob

Bob should send two \( F \)-values per item, what if he sends only one?

Alice has \( c \); does she include it in output?

Only if \( c \) placed in bin 3!

\( \Rightarrow \) depends on Alice's entire input!
why isn’t it secure against malicious parties?

Alice

1  $F_1(\cdot)$
2  $F_2(\cdot)$
3  $F_3(\cdot)$
4  $F_4(\cdot)$
5  $F_5(\cdot)$
6  $F_6(\cdot)$
7  $F_7(\cdot)$
8  $F_8(\cdot)$
9  $F_9(\cdot)$
10 $F_{10}(\cdot)$

Bob

Bob should send two $F$-values per item, what if he sends only one?

Alice has $c$; does she include it in output?

Only if $c$ placed in bin 3!

$\Rightarrow$ depends on Alice’s entire input!

$\{F_3(c), F_3(e), F_4(d), \ldots, F_7(c), \ldots\}$
why isn’t it secure against malicious parties?

Bob should send two $F$-values per item, what if he sends only one?

Alice has $c$; does she include it in output?

$\{F_3(c), F_3(e), F_4(d), \ldots, F_7(c), \ldots\}$
why isn’t it secure against malicious parties?

Bob should send two $F$-values per item, what if he sends only one?

Alice has $c$; does she include it in output?

$\{F_3(c), F_3(e), F_4(d), \ldots, F_7(c), \ldots\}$
why isn’t it secure against malicious parties?

Bob should send two $F$-values per item, what if he sends only one? Alice has $c$; does she include it in output?

Only if $c$ placed in bin 3! Depends on Alice’s entire input! ⇒ can’t simulate!
why isn’t it secure against malicious parties?

Bob should send two $F$-values per item, what if he sends only one?

Alice has $c$; does she include it in output?

Only if $c$ placed in bin 3!

⇒ depends on Alice’s entire input!

$\{F_3(c), F_3(e), F_4(d), \ldots, F_7(c), \ldots\}$
why isn’t it secure against malicious parties?

Alice

Bob should send two $F$-values per item, what if he sends only one?

Alice has $c$; does she include it in output?

Only if $c$ placed in bin 3!
why isn’t it secure against malicious parties?

Bob should send two $F$-values per item, what if he sends only one?

Alice has $c$; does she include it in output?

Only if $c$ placed in bin 3!

▶ Depends on Alice’s entire input!

⇒ can’t simulate!
how do we overcome this problem?
**batch OPRF for malicious PSI**

<table>
<thead>
<tr>
<th>Alice</th>
<th>Bob</th>
</tr>
</thead>
<tbody>
<tr>
<td>$F_1(x_1)$</td>
<td>1</td>
</tr>
<tr>
<td>$F_2(x_2)$</td>
<td>2</td>
</tr>
<tr>
<td>$F_3(x_3)$</td>
<td>3</td>
</tr>
<tr>
<td>$F_4(x_4)$</td>
<td>4</td>
</tr>
<tr>
<td>$F_5(x_5)$</td>
<td>5</td>
</tr>
<tr>
<td>$F_6(x_6)$</td>
<td>6</td>
</tr>
<tr>
<td>$F_7(x_7)$</td>
<td>7</td>
</tr>
<tr>
<td>$F_8(x_8)$</td>
<td>8</td>
</tr>
<tr>
<td>$F_9(x_9)$</td>
<td>9</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
### batch OPRF for malicious PSI

<table>
<thead>
<tr>
<th>Alice</th>
<th>Bob</th>
</tr>
</thead>
<tbody>
<tr>
<td>$F_1(x_1)$</td>
<td>$F_1(\cdot)$</td>
</tr>
<tr>
<td>$F_2(x_2)$</td>
<td>$F_2(\cdot)$</td>
</tr>
<tr>
<td>$F_3(x_3)$</td>
<td>$F_3(\cdot)$</td>
</tr>
<tr>
<td>$F_4(x_4)$</td>
<td>$F_4(\cdot)$</td>
</tr>
<tr>
<td>$F_5(x_5)$</td>
<td>$F_5(\cdot)$</td>
</tr>
<tr>
<td>$F_6(x_6)$</td>
<td>$F_6(\cdot)$</td>
</tr>
<tr>
<td>$F_7(x_7)$</td>
<td>$F_7(\cdot)$</td>
</tr>
<tr>
<td>$F_8(x_8)$</td>
<td>$F_8(\cdot)$</td>
</tr>
<tr>
<td>$F_9(x_9)$</td>
<td>$F_9(\cdot)$</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

State of the art malicious batch OPRF [OOS17]

- essentially same cost as semi-honest
**batch OPRF for malicious PSI**

<table>
<thead>
<tr>
<th>Alice</th>
<th>Bob</th>
</tr>
</thead>
<tbody>
<tr>
<td>$F_1(x_1)$</td>
<td>$F_1(\cdot)$</td>
</tr>
<tr>
<td>$F_2(x_2)$</td>
<td>$F_2(\cdot)$</td>
</tr>
<tr>
<td>$F_3(x_3)$</td>
<td>$F_3(\cdot)$</td>
</tr>
<tr>
<td>$F_4(x_4)$</td>
<td>$F_4(\cdot)$</td>
</tr>
<tr>
<td>$F_5(x_5)$</td>
<td>$F_5(\cdot)$</td>
</tr>
<tr>
<td>$F_6(x_6)$</td>
<td>$F_6(\cdot)$</td>
</tr>
<tr>
<td>$F_7(x_7)$</td>
<td>$F_7(\cdot)$</td>
</tr>
<tr>
<td>$F_8(x_8)$</td>
<td>$F_8(\cdot)$</td>
</tr>
<tr>
<td>$F_9(x_9)$</td>
<td>$F_9(\cdot)$</td>
</tr>
</tbody>
</table>

State of the art malicious batch OPRF [OOS17]

- essentially same cost as semi-honest
- consistency check relies on an additive homomorphism:

$$F_i(x) \oplus F_j(y) = F_{ij}(x \oplus y)$$

*: a gross oversimplification
our protocol main idea:

Alice

a
b
c
d

1
2
3
4
5
6
7
8
9
10

Bob

c
d
e
f

F_1(s_1) \oplus F_2(s_2) = F_2(s_2) \oplus F_7(s_7) = F_2(s_2) \oplus F_3(s_3) = F_3(s_3) \oplus F_9(s_9) = F_3(s_3) \oplus F_7(s_7) = F_4(s_4) \oplus F_7(s_7) = F_4(s_4) \oplus F_7(s_7) = F_10(s_10)

Alice secret-shares x into bins h_1(x) and h_2(x)

Bob sends only one F-value per item
our protocol main idea:

Alice secret-shares $x$ into bins $h_1(x)$ and $h_2(x)$.

Bob sends only one $F$-value per item.
Our protocol main idea:

Alice secret-shares $x$ into bins $h_1(x)$ and $h_2(x)$.
our protocol main idea:

Alice secret-shares $x$ into bins $h_1(x)$ and $h_2(x)$
our protocol main idea:

Alice secret-shares $x$ into bins $h_1(x)$ and $h_2(x)$
our protocol main idea:

Alice secret-shares $x$ into bins $h_1(x)$ and $h_2(x)$.
our protocol main idea:

Alice secret-shares $x$ into bins $h_1(x)$ and $h_2(x)$.
our protocol main idea:

Alice secret-shares $x$ into bins $h_1(x)$ and $h_2(x)$
our protocol main idea:

Alice secret-shares $x$ into bins $h_1(x)$ and $h_2(x)$
our protocol main idea:

Alice secret-shares $x$ into bins $h_1(x)$ and $h_2(x)$

<table>
<thead>
<tr>
<th>Alice</th>
<th>Bob</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_2 \oplus s_7 = a$</td>
<td>$F_1(s_1)$</td>
</tr>
<tr>
<td>$s_3 \oplus s_9 = b$</td>
<td>$F_2(s_2)$</td>
</tr>
<tr>
<td>$s_3 \oplus s_7 = c$</td>
<td>$F_3(s_3)$</td>
</tr>
<tr>
<td>$s_4 \oplus s_7 = d$</td>
<td>$F_4(s_4)$</td>
</tr>
<tr>
<td>$F_2(s_2) \oplus F_7(s_7) = F_{27}(a)$</td>
<td></td>
</tr>
<tr>
<td>$F_3(s_3) \oplus F_9(s_9) = F_{39}(b)$</td>
<td></td>
</tr>
<tr>
<td>$F_3(s_3) \oplus F_7(s_7) = F_{37}(c)$</td>
<td></td>
</tr>
<tr>
<td>$F_4(s_4) \oplus F_7(s_7) = F_{47}(d)$</td>
<td></td>
</tr>
</tbody>
</table>
**our protocol main idea:**

Alice secret-shares $x$ into bins $h_1(x)$ and $h_2(x)$

<table>
<thead>
<tr>
<th>Alice</th>
<th>Bob</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_2 \oplus s_7 = a$</td>
<td>$F_1(s_1)$, $F_1(\cdot)$</td>
</tr>
<tr>
<td>$s_3 \oplus s_9 = b$</td>
<td>$F_2(s_2)$, $F_2(\cdot)$</td>
</tr>
<tr>
<td>$s_3 \oplus s_7 = c$</td>
<td>$F_3(s_3)$, $F_3(\cdot)$</td>
</tr>
<tr>
<td>$s_4 \oplus s_7 = d$</td>
<td>$F_4(s_4)$, $F_4(\cdot)$</td>
</tr>
<tr>
<td>$F_2(s_2) \oplus F_7(s_7) = F_{27}(a)$</td>
<td></td>
</tr>
<tr>
<td>$F_3(s_3) \oplus F_9(s_9) = F_{39}(b)$</td>
<td></td>
</tr>
<tr>
<td>$F_3(s_3) \oplus F_7(s_7) = F_{37}(c)$</td>
<td></td>
</tr>
<tr>
<td>$F_4(s_4) \oplus F_7(s_7) = F_{47}(d)$</td>
<td></td>
</tr>
</tbody>
</table>
our protocol main idea:

Alice secret-shares $x$ into bins $h_1(x)$ and $h_2(x)$.

$F_2(s_2) \oplus F_7(s_7) = F_{27}(a)$
$F_3(s_3) \oplus F_9(s_9) = F_{39}(b)$
$F_3(s_3) \oplus F_7(s_7) = F_{37}(c)$
$F_4(s_4) \oplus F_7(s_7) = F_{47}(d)$

Alice

$s_2 \oplus s_7 = a$
$s_3 \oplus s_9 = b$
$s_3 \oplus s_7 = c$
$s_4 \oplus s_7 = d$

Bob

$F_1(s_1)$
$F_2(s_2)$
$F_3(s_3)$
$F_4(s_4)$
$F_5(s_5)$
$F_6(s_6)$
$F_7(s_7)$
$F_8(s_8)$
$F_9(s_9)$
$F_{10}(s_{10})$

1
2
3
4
5
6
7
8
9
10

$F_1(\cdot)$
$F_2(\cdot)$
$F_3(\cdot)$
$F_4(\cdot)$
$F_5(\cdot)$
$F_6(\cdot)$
$F_7(\cdot)$
$F_8(\cdot)$
$F_9(\cdot)$
$F_{10}(\cdot)$

$F_{37}(c)$,
our protocol main idea:

Alice secret-shares $x$ into bins $h_1(x)$ and $h_2(x)$

$F_2(s_2) \oplus F_7(s_7) = F_{27}(a)$
$F_3(s_3) \oplus F_9(s_9) = F_{39}(b)$
$F_3(s_3) \oplus F_7(s_7) = F_{37}(c)$
$F_4(s_4) \oplus F_7(s_7) = F_{47}(d)$
**our protocol main idea:**

<table>
<thead>
<tr>
<th>Alice</th>
<th></th>
<th>Bob</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_2 \oplus s_7 = a$</td>
<td>$F_1(s_1)$</td>
<td>$F_1(\cdot)$</td>
</tr>
<tr>
<td>$s_3 \oplus s_9 = b$</td>
<td>$F_2(s_2)$</td>
<td>$F_2(\cdot)$</td>
</tr>
<tr>
<td>$s_3 \oplus s_7 = c$</td>
<td>$F_3(s_3)$</td>
<td>$F_3(\cdot)$</td>
</tr>
<tr>
<td>$s_4 \oplus s_7 = d$</td>
<td>$F_4(s_4)$</td>
<td>$F_4(\cdot)$</td>
</tr>
<tr>
<td>$F_2(s_2) \oplus F_7(s_7) = F_{27}(a)$</td>
<td>$F_5(s_5)$</td>
<td>$F_5(\cdot)$</td>
</tr>
<tr>
<td>$F_3(s_3) \oplus F_9(s_9) = F_{39}(b)$</td>
<td>$F_6(s_6)$</td>
<td>$F_6(\cdot)$</td>
</tr>
<tr>
<td>$F_3(s_3) \oplus F_7(s_7) = F_{37}(c)$</td>
<td>$F_7(s_7)$</td>
<td>$F_7(\cdot)$</td>
</tr>
<tr>
<td>$F_4(s_4) \oplus F_7(s_7) = F_{47}(d)$</td>
<td>$F_8(s_8)$</td>
<td>$F_8(\cdot)$</td>
</tr>
<tr>
<td>$F_9(s_9)$</td>
<td>$F_9(\cdot)$</td>
<td></td>
</tr>
<tr>
<td>$F_{10}(s_{10})$</td>
<td>$F_{10}(\cdot)$</td>
<td></td>
</tr>
</tbody>
</table>

Alice secret-shares $x$ into bins $h_1(x)$ and $h_2(x)$

Bob sends only one $F$-value per item

\{ $F_{37}(c), F_{47}(d), F_{35}(e), F_{69}(f) \}$
our protocol main idea:

Alice secret-shares $x$ into bins $h_1(x)$ and $h_2(x)$

Bob sends only one $F$-value per item
[how] can Alice secret-share all items?

\[\begin{align*}
  s_2 \oplus s_7 &= a \\
  s_3 \oplus s_9 &= b \\
  s_3 \oplus s_7 &= c \\
  s_4 \oplus s_7 &= d
\end{align*}\]
[how] can Alice secret-share all items?

\[
s_2 \oplus s_7 = a \quad s_3 \oplus s_9 = b \quad s_3 \oplus s_7 = c \quad s_4 \oplus s_7 = d
\]

cuckoo graph

\[
\text{set one location arbitrarily}
\]

\[
\text{repeat:}
\]

\[
\text{/f_ind item with one unset location}
\]

\[
\text{solve for that unset location}
\]

\[
\text{only works if cuckoo graph acyclic}
\]
[how] can Alice secret-share all items?

ALGORITHM:
set one location arbitrarily
[how] can Alice secret-share all items?

\[
s_2 \oplus s_7 = a
\]
\[
s_3 \oplus s_9 = b
\]
\[
s_3 \oplus s_7 = c
\]
\[
s_4 \oplus s_7 = d
\]

ALGORITHM:
set one location arbitrarily

find item with one unset location
[how] can Alice secret-share all items?

**ALGORITHM:**

set one location arbitrarily

find item with one unset location

solve for that unset location

$s_2 \oplus s_7 = a$

$s_3 \oplus s_9 = b$

$s_3 \oplus s_7 = c$

$s_4 \oplus s_7 = d$
[how] can Alice secret-share all items?

ALGORITHM:
set one location arbitrarily
repeat:
  find item with one unset location
  solve for that unset location
[how] can Alice secret-share all items?

```
s_2 \oplus s_7 = a
s_3 \oplus s_9 = b
s_3 \oplus s_7 = c
s_4 \oplus s_7 = d
```

**Algorithm:**
- set one location arbitrarily
- repeat:
  - find item with one unset location
  - solve for that unset location

**Cuckoo Graph:**
- Set one location arbitrarily
- Repeat:
  - Find item with one unset location
  - Solve for that unset location

- Only works if cuckoo graph is acyclic.
[how] can Alice secret-share all items?

\[
s_2 \oplus s_7 = a
\]
\[
s_3 \oplus s_9 = b
\]
\[
s_3 \oplus s_7 = c
\]
\[
s_4 \oplus s_7 = d
\]

**ALGORITHM:**

set one location arbitrarily
repeat:
  find item with one unset location
  solve for that unset location
[how] can Alice secret-share all items?

\[
\begin{align*}
    s_2 \oplus s_7 &= a \\
    s_3 \oplus s_9 &= b \\
    s_3 \oplus s_7 &= c \\
    s_4 \oplus s_7 &= d
\end{align*}
\]

ALGORITHM:

set one location arbitrarily
repeat:
    find item with one unset location
    solve for that unset location
[how] can Alice secret-share all items?

\[
s_2 \oplus s_7 = a
\]
\[
s_3 \oplus s_9 = b
\]
\[
s_3 \oplus s_7 = c
\]
\[
s_4 \oplus s_7 = d
\]

**ALGORITHM:**

set one location arbitrarily
repeat:
    find item with one unset location
    solve for that unset location
[how] can Alice secret-share all items?

**ALGORITHM:**
- set one location arbitrarily
- repeat:
  - find item with one unset location
  - solve for that unset location

\[
s_2 \oplus s_7 = a \\
s_3 \oplus s_9 = b \\
s_3 \oplus s_7 = c \\
s_4 \oplus s_7 = d
\]
[how] can Alice secret-share all items?

**ALGORITHM:**

set one location arbitrarily
repeat:
  find item with one unset location
  solve for that unset location
[how] can Alice secret-share all items?

ALGORITHM:
set one location arbitrarily
repeat:
  find item with one unset location
  solve for that unset location
only works if cuckoo graph acyclic

cuckoo graph
encode so that for all $x$:

$$s_1(x) \oplus s_2(x) = x$$

1. system of linear constraints must be satisfiable with overwhelming probability

2. $|\mathcal{S}| = \text{number of OPRFs} = \text{communication cost of PSI, ideally } O(\# \text{ items})$

3. ideally linear-time encoding of items into $\mathcal{S}$. 

Diagram:

- $s_2 \oplus s_7 = a \rightarrow s_2 \rightarrow s_1$
- $s_3 \oplus s_9 = b \rightarrow s_3 \rightarrow s_2$ and $s_5 \rightarrow s_4$
- $s_3 \oplus s_7 = c \rightarrow s_4 \rightarrow s_3$ and $s_6 \rightarrow s_5$
- $s_4 \oplus s_7 = d \rightarrow s_7 \rightarrow s_4$ and $s_8 \rightarrow s_6$ and $s_9 \rightarrow s_7$
encode so that for all $x$:

$$S_{h_1}(x) \oplus S_{h_2}(x) = x$$
probe-and-XOR-of-strings (PaXoS)

Encode so that for all $x$: $\bigoplus_{i \in P(x)} s_i = x$

1. System of linear constraints must be satisfiable with overwhelming probability.
2. $|\mathcal{S}| =$ number of OPRFs = communication cost of PSI, ideally $O(\# \text{items})$.
3. Ideally linear-time encoding of items into $\mathcal{S}$.
**probe-and-XOR-of-strings (PaXoS)**

Encode so that for all $x$:

$$\bigoplus_{i\in P(x)} s_i = x$$

1. system of linear constraints must be **satisfiable** with overwhelming probability.
probe-and-XOR-of-strings (PaXoS)

1. system of linear constraints must be satisfiable with overwhelming probability
2. $|\mathcal{S}| = \text{number of OPRFs} = \text{communication cost of PSI, ideally } O(\# \text{ items})$
probe-and-XOR-of-strings (PaXoS)

1. system of linear constraints must be satisfiable with overwhelming probability
2. $|\vec{s}| =$ number of OPRFs = communication cost of PSI, ideally $O(\# \text{ items})$
3. ideally linear-time encoding of items into $\vec{s}$.

Encode so that for all $x$:

$$\bigoplus_{i \in P(x)} s_i = x$$
PaXoS constructions

secret-shared cuckoo idea:

- requires acyclic cuckoo graph
- failure probability too high
PaXoS constructions

secret-shared cuckoo idea:
- requires acyclic cuckoo graph
- failure probability too high

probe each position with probability 0.5:
- $n$ items $\leadsto$ vector of size $n + \lambda$
- expensive $O(n^3)$ encoding
PaXoS constructions

secret-shared cuckoo idea:
  ▶ requires acyclic cuckoo graph
  ▶ failure probability too high

probe each position with probability 0.5:
  ▶ $n$ items $\sim$ vector of size $n + \lambda$
  ▶ expensive $O(n^3)$ encoding

garbled bloom filter [DCW13]:
  ▶ $n$ items $\sim$ vector of size $\lambda n$
**PaXoS constructions**

secret-shared cuckoo idea:
- requires acyclic cuckoo graph
- **failure probability** too high

probe each position with probability 0.5:
- $n$ items $\sim$ vector of size $n + \lambda$
- expensive $O(n^3)$ encoding

garbled bloom filter [DCW13]:
- $n$ items $\sim$ vector of size $\lambda n$

**new garbled cuckoo PaXoS:**
- $n$ items $\sim$ vector of size $\sim 2.4n$
- fast encoding: $O(n\lambda)$
new garbled cuckoo PaXoS

for each item $x$:
- probe positions $h_1(x)$ and $h_2(x)$

for each item $x$:
- probe positions $h_1(x)$ and $h_2(x)$
new garbled cuckoo PaXoS

for each item $x$:
  - probe positions $h_1(x)$ and $h_2(x)$

[Diagram]

- $s_1$ to $s_{10}$
- Nodes labeled a, b, c, d, e
- Arrows indicating connections

- $k$ aux positions

1. identify all items across all cycles
   - solve linear system for the cycle items
   - solution exists whp if $k > [\# cycle items] + \lambda$

2. solve for remaining items iteratively (linear time)
   - remaining cuckoo graph is acyclic
     - whp: $[\# cycle items] = O(\log n)$
new garbled cuckoo PaXoS

for each item $x$:

- probe positions $h_1(x)$ and $h_2(x)$

- probe random subset of aux positions

1. identify all items across all cycles

- solve linear system for the cycle items

- solution exists whp if $k > \lceil \#\text{cycle items} \rceil + \lambda$

- can be found in $\lceil \#\text{cycle items} \rceil^3$ time

2. solve for remaining items iteratively (linear time)

- remaining cuckoo graph is acyclic whp: $\lceil \#\text{cycle items} \rceil = O(\log n)$
new garbled cuckoo PaXoS

for each item $x$:
- probe positions $h_1(x)$ and $h_2(x)$
- probe random subset of aux positions

for each item $x$:
- probe positions $h_1(x)$ and $h_2(x)$
- probe random subset of aux positions

$k$ aux positions
new garbled cuckoo PaXoS

for each item $x$:
  ▶ probe positions $h_1(x)$ and $h_2(x)$
  ▶ probe random subset of aux positions
new garbled cuckoo PaXoS

for each item $x$:
  ▶ probe positions $h_1(x)$ and $h_2(x)$
  ▶ probe random subset of aux positions

1. identify all items across all cycles
new garbled cuckoo PaXoS

for each item \( x \):

- probe positions \( h_1(x) \) and \( h_2(x) \)
- probe random subset of aux positions

1. identify all items across all cycles
   - solve linear system for the cycle items

\( \# \) cycle items is \( O(\log n) \)
new garbled cuckoo PaXoS

for each item $x$:
- probe positions $h_1(x)$ and $h_2(x)$
- probe random subset of aux positions

1. identify all items across all cycles
   - solve linear system for the cycle items
   - solution exists whp if $k > \lceil \# \text{cycle items} \rceil + \lambda$
new garbled cuckoo PaXoS

for each item $x$:
- probe positions $h_1(x)$ and $h_2(x)$
- probe random subset of aux positions

1. identify all items across all cycles
- solve linear system for the cycle items
- solution exists whp if $k > \lceil \# \text{cycle items} \rceil + \lambda$
- can be found in $\lceil \# \text{cycle items} \rceil^3$ time
new garbled cuckoo PaXoS

for each item $x$:
  - probe positions $h_1(x)$ and $h_2(x)$
  - probe random subset of aux positions

1. identify all items across all cycles
  - solve linear system for the cycle items
  - solution exists whp if $k > \lceil \# \text{cycle items} \rceil + \lambda$
  - can be found in $\lceil \# \text{cycle items} \rceil^3$ time

2. solve for remaining items iteratively (linear time)
new garbled cuckoo PaXoS

for each item $x$:
- probe positions $h_1(x)$ and $h_2(x)$
- probe random subset of aux positions

1. identify all items across all cycles
   - solve linear system for the cycle items
   - solution exists whp if $k > [\# \text{ cycle items}] + \lambda$
   - can be found in $[\# \text{ cycle items}]^3$ time

2. solve for remaining items \textit{iteratively} (linear time)
new garbled cuckoo PaXoS

for each item $x$:
- probe positions $h_1(x)$ and $h_2(x)$
- probe random subset of aux positions

1. identify all items across all cycles
   - solve linear system for the cycle items
   - solution exists whp if $k > \lceil \# \text{ cycle items} \rceil + \lambda$
   - can be found in $\lceil \# \text{ cycle items} \rceil^3$ time

2. solve for remaining items iteratively (linear time)
   - remaining cuckoo graph is acyclic
new garbled cuckoo PaXoS

for each item $x$:
  - probe positions $h_1(x)$ and $h_2(x)$
  - probe random subset of aux positions

1. identify all items across all cycles
   - solve linear system for the cycle items
   - solution exists whp if $k > \lceil \text{# cycle items} \rceil + \lambda$
   - can be found in $\lceil \text{# cycle items} \rceil^3$ time

2. solve for remaining items **iteratively** (linear time)
   - remaining cuckoo graph is acyclic
new garbled cuckoo PaXoS

for each item $x$:
- probe positions $h_1(x)$ and $h_2(x)$
- probe random subset of aux positions

1. identify all items across all cycles
   - solve linear system for the cycle items
   - solution exists whp if $k > \lceil \# \text{cycle items} \rceil + \lambda$
   - can be found in $\lceil \# \text{cycle items} \rceil^3$ time

2. solve for remaining items \textit{iteratively} (linear time)
   - remaining cuckoo graph is acyclic
new garbled cuckoo PaXoS

for each item $x$:
- probe positions $h_1(x)$ and $h_2(x)$
- probe random subset of aux positions

1. identify all items across all cycles
- solve linear system for the cycle items
- solution exists whp if $k > \lceil \#\text{cycle items} \rceil + \lambda$
- can be found in $\lceil \#\text{cycle items} \rceil^3$ time

2. solve for remaining items **iteratively** (linear time)
- remaining cuckoo graph is acyclic
**new garbled cuckoo PaXoS**

For each item $x$:
- probe positions $h_1(x)$ and $h_2(x)$
- probe random subset of aux positions

1. Identify all items across all cycles
   - probe positions $h_1(x)$ and $h_2(x)$
   - solution exists whp if $k > \lceil \# \text{cycle items} \rceil + \lambda$
   - can be found in $\lceil \# \text{cycle items} \rceil^3$ time

2. Solve for remaining items iteratively (linear time)
   - remaining cuckoo graph is acyclic

whp: $\lceil \# \text{cycle items} \rceil$ is $O(\log n)$
new approach for malicious PSI:
- fastest, least communication
- first $O(n)$ from OT extension
- almost as efficient as semi-honest

PaXoS data structure:
- encode items into a vector
- lookup is XOR of some positions
- first linear-time, constant rate construction
thanks!