Improved Private Set Intersection against Malicious Adversaries
Private Set Intersection (PSI)

\[
X \cap Y
\]
Private Set Intersection (PSI)

\[ X \cap Y \]
App: Contact discovery

Users \( \rightarrow \) PSI \( \rightarrow \) Contacts

\( X \cap Y \)
Oblivious Transfer (OT)

- Highly efficient and secure protocols exist.
- Motivates its use as the basis for PSI.
Bloom Filter

Plain text data structure similar to hash table

- Allows for testing set membership
- Parameterized by hash functions $h_1, \ldots, h_k$

- \[ b[h_i(x)] = 1 \quad \forall i \]

\[ b = \begin{array}{cccccccccccccccc}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{array} \]
Bloom Filter

Plain $i(x) = 1$, $\forall i$

ext data structure similar to hash table
  - Allows for testing set membership
  - Parameterized by hash functions $h_1, \ldots, h_k$

• To insert $x$, set
  - $b_{h_i h_i(x)} = 1$, $\forall i$

$$b = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$
Bloom Filter

Plain \( i(x) = 1 \), \( \forall i \)

ext data structure similar to hash table
  - Allows for testing set membership
  - Parameterized by hash functions \( h_1, \ldots, h_k \)

• To insert \( x \), set
  • \( b \ h \ i \ h \ i (x) = 1 \), \( \forall i \)

\[
\begin{array}{c}
\text{b} = \\
0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 1
\end{array}
\]
Bloom Filter

• Plain text data structure similar to hash table
  • Allows for testing set membership
  • Parameterized by hash functions $h_1, ..., h_n$

• To insert $x$, set
  • $b[h_i(x)] = 1$, $\forall i$

• To test membership
  • Return $\land_i b[h_i(x)]$

$b = \begin{array}{ccccccccccccccc}
0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 1
\end{array}$
Bloom Filter

- Plain text data structure similar to hash table
  - Allows for testing set membership
  - Parameterized by hash functions $h_1, \ldots, h_n$

- To insert $x$, set
  - $b[h_i(x)] = 1$, $\forall i$

- To test membership
  - Return $\land_i b[h_i(x)]$

\[ b = \begin{array}{cccccccccccc}
0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 1
\end{array} \]
Bloom Filter

\( n \) items → Bloom filter with \( m \) slots and \( k \) hash functions

- Membership:
  - \( \Pr[\text{false negatives}] = 0 \)
  - \( (1 - e^{-kn/m})^k \)
    \[ \approx 2^{-k} \]

\[
\begin{array}{cccccccccccccccccccccc}
0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 1
\end{array}
\]

\( h_1(x) \) \hspace{1cm} h_2(x) \hspace{1cm} \ldots \hspace{1cm} h_k(x) \)
Bloom Filter

A Bloom filter with \( m \) slots and \( k \) hash functions

- Membership:
  - \( \Pr[\text{false negatives}] = 0 \)
  - \( \Pr[\text{false positives}] = (1 - e^{-kn/m})^k \approx 2^{-k} \)

\[ b = \begin{array}{cccccccccccccc}
0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 1
\end{array} \]
Bloom Filter

\[ \approx 2 - k \ 2 - k \ 2 - k \]
\[ \approx 2 - k \ 2 - k \ 2 - k \]

\( k \ n \ m \ k k n n \ k n \ m \ m m \ k n \ m \ e - k n \ m \ 1 - e - k n \ m \ 1 - e - k n \ m \ k k \ 1 - e - k n \ m \ k \ e - k n \ m \ e e e - k n \ m \ --- 1 - e - k n \ m \ k \ 1 - e - k n \ m \ 1 n \ \text{items} \rightarrow \text{Bloom filter with } m \text{ slots and } k \text{ hash functions} \)

- Membership:
  \[ \approx 2^{-k} \approx 2^{-k} \]

\[ b = \begin{array}{ccccccccccccccccc}
0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 \\
\end{array} \]
Bloom Filter Intersection

- Bitwise AND $B_X \wedge B_Y$ is a Bloom filter for $X \cap Y$

$X = \{a, b\}$

$B_X$

$h_i(a)$

$h_i(b)$

$Y = \{a, c\}$

$B_Y$
Bloom Filter Intersection

- Bitwise AND $B_X \land B_Y$ is a Bloom filter for $X \cap Y$

$X = \{a, b\}$

$B_X$

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

$B_X \land B_Y$

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

$Y = \{a, c\}$

$B_Y$

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

$h_i(a)$

$h_i(b)$

$h_i(a)$

$h_i(c)$
Bloom Filter Protocol

$m_i \leftarrow \{0, 1\}^\kappa$

$B_Y$

$Y = \{a, c\}$

$h_i(a)$

$h_i(c)$
Bloom Filter Protocol

$m_i \leftarrow \{0, 1\}^\kappa$

[ DongChenWen13, PinkasSchniederZohner14 ]
Bloom Filter Protocol

Garbled Bloom filter

\[ B_Y \]

\[ Y = \{a, c\} \]

[Garbled Bloom filter]

[DongChenWen13, PinkasSchniederZohner14]
Bloom Filter Protocol

\[ X = \{ a, b \} \]

\[ B_X \]

\[
\begin{array}{c|c}
\hline
& m_0 & m_1 & m_2 & m_3 & m_4 & m_5 & m_6 \\
\hline
\hline
1 & \perp & \perp & 1 & \perp & \perp & 1 & \perp \\
\hline
\end{array}
\]

\[ X = \{ a, b \} \]

\[ B_Y \]

\[
\begin{array}{c|c}
\hline
& m_0 & m_1 & m_2 & m_3 & m_4 & m_5 & m_6 \\
\hline
\hline
1 & \perp & \perp & \perp & 1 & \perp & 1 & \perp \\
\hline
\end{array}
\]

Garbled Bloom filter

[DongChenWen13, PinkasSchniederZohner14]
Bloom Filter Protocol

$X = \{a, b\}$

$B_X$

$B_Y$

$Y = \{a, c\}$

$\hat{X} = \{m_0 \oplus m_5, m_2 \oplus m_3\}$

Garbled Bloom filter

[DongChenWen13, PinkasSchniederZohner14]
Bloom Filter Protocol

\(X = \{a, b\}\)

\(B_X\)

\(h_i(a)\)

\(h_i(b)\)

\(\widehat{X} = \{m_0 \oplus m_5, m_2 \oplus m_3\}\)

Garbled Bloom filter

\(Y = \{a, c\}\)

\(B_Y\)

\(h_i(a)\)

\(h_i(c)\)

Output the intersection

\(\widehat{X} \cap \{m_0 \oplus m_5, m_3 \oplus m_6\}\)
Semi-Honest Security

Naturally secure against Sender:

- OT hides select bits
- Final message sent to Receiver

- \( \notin Y \), Receiver learns encoding
e.g. \( \text{Encode}(y') = m_3 \oplus m_4 \)

- DCW13 show equivalence to false positive in standard bloom filter
  - \( \Pr[\text{false positives}] \approx 2^{-k} \)

\[
\begin{align*}
X &= \{a, b\} \\
Y &= \{a, c\} \\
\hat{X} &= \{m_0 \oplus m_5, m_2 \oplus m_3\} \\
\hat{X} \cap \{m_0 \oplus m_5, m_3 \oplus m_6\}
\end{align*}
\]
Semi-Honest Security

[DongChenWen13, PinkasSchniederZohner14]

\( y' \notin YY \), Receiver learns encoding

Naturally secure against Sender:

- OT hides select bits
- Final message sent to Receiver

- Secure against Receiver
  - Attack: For \( y'' \notin Y \), Receiver learns encoding
    
    e.g. \( \text{Encode}(y') = m_3 \oplus m_4 \)

- DCW13 show equivalence to false positive in standard bloom filter
  - \( \Pr[\text{false positives}] \approx 2^{-k} \)
Semi-Honest Security

\[ m_0 \oplus m_3 \]
\[ m_1 \]
\[ m_2 \]
\[ \vdots \]
\[ m_5 \]
\[ m_6 \]

\[ X = \{a, b\} \]
\[ Y = \{a, c\} \]

\[ Y = \{a, c\} \]
\[ h_i(a) \]
\[ h_i(c) \]

\[ \text{Output:} \]
\[ \hat{X} = \{m_0 \oplus m_5, m_2 \oplus m_3\} \]
\[ \hat{X} \cap \{m_0 \oplus m_5, m_3 \oplus m_6\} \]

\[ \text{e.g.}\]
\[ \text{Encode } y' y y' = m \ 3 \ m \ 3 \ 3 \ m \ 3 \ \oplus \]
\[ m \ 4 \ m \ 4 \ m \ 4 \ m \ 4 \]

\[ y' \notin YY, \text{ Receiver learns encoding} \]

Naturally secure against Sender.
- OT hides select bits
- Final message sent to Receiver

- Secure against Receiver
  \[ \text{ncode}(y') = m_3 \oplus m_4 \text{ e.g.} \]
  \[ \text{Encode}(y') = m_3 \oplus m_4 \]

- DCW13 show equivalence to false positive in standard bloom filter
  \[ \text{Pr[ false positives ]} \approx 2^{-k} \]
Semi-Honest Security

\[ Y = \{a, c\} \]

\[ \begin{array}{c|c}
\hline
0 & m_0 \\
1 & m_1 \\
0 & m_2 \\
1 & m_3 \\
0 & m_4 \\
1 & m_5 \\
0 & m_6 \\
\hline
\end{array} \]

\[ h_i(a) \]
\[ h_i(c) \]

\[ \begin{array}{c|c}
\hline
0 & m_0 \\
1 & m_1 \\
0 & m_2 \\
1 & m_3 \\
0 & m_4 \\
1 & m_5 \\
0 & m_6 \\
\hline
\end{array} \]

\[ h_i(a) \]
\[ h_i(c) \]

Output:

\[ \hat{X} = \{m_0 \oplus m_5, m_2 \oplus m_3\} \]

\[ \hat{X} \cap \{m_0 \oplus m_5, m_3 \oplus m_6\} \]

e.g. Encode \( y'\) \( y'y' = m_3 m_4 m_3 m_3 \oplus m_4 m_4 m_4 m_4 \)

\( y' \notin YY \), Receiver learns encoding

Naturally secure against Sender.

- OT hides select bits
- Final message sent to Receiver

Secure against Receiver

\[ \text{ncode}(y') = m_3 \oplus m_4 \]

\[ \text{Encode}(y') = m_3 \oplus m_4 \]

e.g.

- DCW13 show equivalence to false positive in standard bloom filter
- DCW13 show equivalence to false positive in standard bloom filter
  - \( \Pr[\text{false positives}] \approx 2^{-k} \)
Semi-Honest Security

2 \kappa 2 2 \kappa 2 \kappa 2 \kappa

e.g. Encode \( y' yy' y' = m_3 m_3 m_3 m_4 m_4 m_4 m_4 m_4 m_4 m_4 \)

\( y' \notin YY \), Receiver learns encoding

Naturally secure against Sender.
- OT hides select bits
- Final message sent to Receiver

- Secure against Receiver
  \( \text{encode}(y') = m_3 \oplus m_4 \)
e.g. Encode \( y' = m_3 \oplus m_4 \)

- DCW13 show equivalence to false positive in standard bloom filter
  - \( \Pr[\text{false positives}] \approx 2^{-k} \)
  - \( \Pr[\text{false positives}] \approx 2^{-k} \)
Malicious Receiver

**Insecure** against Receiver

- Bloom filter
- Receiver will obtain all \( m_i \)
- Can probe for \( m_2 \oplus m_3 \)

\[
\begin{align*}
X &= \{a, b\} \\
Y &= \{a, c\}
\end{align*}
\]

\[
\begin{array}{c|c}
\downarrow & m_0 \\
\hline
m_1 & m_1 \\
\hline
m_2 & m_2 \\
\hline
\vdots & \vdots \\
\hline
m_4 & m_4 \\
\hline
m_5 & m_5 \\
\hline
\downarrow & m_6 \\
\end{array}
\quad
\begin{array}{c|c|c}
\downarrow & 1 \\
\hline
1 & 1 \\
\hline
\hline
1 & 1 \\
\hline
\end{array}
\]

Output:

\[
\hat{X} = \left\{ m_0 \oplus m_5, m_2 \oplus m_3 \right\}
\]

\[
\hat{X} \cap \left\{ m_0 \oplus m_5, m_3 \oplus m_6 \right\}
\]
Malicious Receiver

Bloom filter

Insecure against Receiver

- Consider all 1 Bloom filter
- Receiver will obtain all $m_i$
- Can probe for $m_2 \oplus m_3$

\[
\begin{align*}
X &= \{a, b\} \\
Y &= \{a, c\}
\end{align*}
\]

\[
\begin{array}{c|c}
    & m_0 & m_1 & m_2 & m_3 & m_4 & m_5 & m_6 \\
\hline
\bot & \bot & m_1 & m_2 & m_3 & m_4 & m_5 & m_6 \\
\end{array}
\]

\[
\begin{array}{c|c}
    & m_0 & m_1 & m_2 & m_3 & m_4 & m_5 & m_6 \\
\hline
\bot & \bot & \bot & \bot & 1 & \bot & 1 & 1 \\
\end{array}
\]

Output:

\[
\begin{align*}
\hat{X} &= \{m_0 \oplus m_5, m_2 \oplus m_3\} \\
\hat{X} \cap \{m_0 \oplus m_5, m_3 \oplus m_6\}
\end{align*}
\]
Malicious Receiver

Bloom filter

Insecure against Receiver

- Receiver will obtain all $m_i$
- $i i i$
- Receiver will obtain all $m_i$
- Can probe for $m_2 \oplus m_3$

\[
X = \{a, b\}
\]
\[
Y = \{a, c\}
\]

\[
\begin{array}{c|c|c|c|c|c|c|c}
\hline
& m_0 & m_1 & m_2 & m_3 & m_4 & m_5 & m_6 \\
\hline
\hline
\bot & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\hline
m_0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
m_1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
m_2 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
m_3 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
m_4 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
m_5 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
m_6 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
\hline
\end{array}
\]

\[
\begin{array}{c|c|c|c|c|c}
\hline
\hline
m_0 \oplus m_5 & 1 \\
m_2 \oplus m_3 & 1 \\
\hline
\end{array}
\]

Output:

\[
\hat{X} = \{m_0 \oplus m_5, m_2 \oplus m_3\}
\]

\[
\hat{Y} \cap \{m_0 \oplus m_5, m_3 \oplus m_6\}
\]
Malicious Receiver

$m \in \{m_0, m_1, m_2, \ldots, m_6\}$

Bloom filter

Insecure against Receiver

- Receiver will obtain all $m \ i$
- Can probe for $m_2 \oplus 2 \ 2 \ 2 \oplus m_3$
- Receiver will obtain all $m_i$
- Can probe for $m_2 \oplus m_3$

\[
X = \{a, b\} \\
Y = \{a, c\} \\
\begin{array}{c|c}
\downarrow & m_0 \\
& m_1 \\
& m_2 \\
& m_3 \\
& m_4 \\
& m_5 \\
\downarrow & m_6 \\
\end{array} \\
\begin{array}{c|c}
& m_0 \\
& m_1 \\
& m_2 \\
& m_3 \\
& m_4 \\
& m_5 \\
& m_6 \\
\end{array}
\]

Output:

\[
\hat{X} = \{m_0 \oplus m_5, m_2 \oplus m_3\} \\
\hat{X} \cap \{m_0 \oplus m_5, m_3 \oplus m_6\}
\]
Warm-Up: The DongChenWen13 Approach

Goal — restrict the Receiver to a valid Bloom filter
- Bits contains \( \frac{1}{2} m \) ones

- Make Receiver prove zero choice bits
  - Sample random key \( s \leftarrow \{0,1\}^\kappa \)
  - Generate a \( \frac{m}{2} \) out of \( m \) secret sharing of \( s \)
    - \( s_1, \ldots, s_m \)

- Transmit \( s_i \) as the \( i \)th zero OT message
- Encrypt summary values with \( s \)
Warm-Up: The DongChenWen13 Approach

1 2 2 1 2 mm ones

Goal — restrict the Receiver to a valid Bloom filter
- Bloom filter of $m$ bits contains 1 2 $m$ ones
- Make Receiver prove zero choice bits
  - Sample random key $s \leftarrow \{0,1\}^\kappa$
  - Generate a $\frac{m}{2}$ out of $m$ secret sharing of $s$
    - $s_1, \ldots, s_m$
  - Transmit $s_i$ as the $i$th zero OT message
  - Encrypt summary values with $s$

\[
X = \{a, b\} \quad \quad Y = \{a, c\}
\]

\[
\begin{array}{c|c|c}
\top & m_0 & m_1 \\
\hline
 & m_1 & m_2 \\
\vdots & m_3 & m_4 \\
\hline
 & m_4 & m_5 \\
\hline
\top & m_6 & m_7 \\
\end{array}
\]

Output:

\[
\hat{X} = \{m_0 \oplus m_5, m_2 \oplus m_3\} \quad \quad \hat{Y} = \{m_0 \oplus m_5, m_3 \oplus m_6\}
\]

\[
h_i(a) \quad h_i(c)
\]
Warm-Up: The DongChenWen13 Approach

0,1 κ 0,1 0,1 0,1 κ κ κ 0,1 κ
1 2 2 1 2 m m ones

Goal — restrict the Receiver to a valid Bloom filter
- Bloom filter of m bits contains 1 2 m ones
- Make Receiver prove zero choice bits
  - Sample random key \( s \leftarrow \{0,1\}^\kappa \)
  - Sample random key \( s \leftarrow \{0,1\}^\kappa \)
  - Generate a \( \frac{m}{2} \) out of m secret sharing of \( s \)
    - \( s_1, \ldots, s_m \)
  - Transmit \( s_i \) as the \( i \)th zero OT message
  - Encrypt summary values with \( s \)

\[
\begin{align*}
Y = \{a, c\} \\
X = \{a, b\}
\end{align*}
\]

Output:

\[
\hat{X} = \{m_0 \oplus m_5, m_2 \oplus m_3\} \\
\hat{X} \cap \{m_0 \oplus m_5, m_3 \oplus m_6\}
\]

\[
\begin{array}{|c|c|}
\hline
m_0 & 1 \\
\hline
m_1 & 1 \\
\hline
m_2 & 1 \\
\hline
m_3 & 1 \\
\hline
m_4 & 1 \\
\hline
m_5 & 1 \\
\hline
m_6 & 1 \\
\hline
\end{array}
\]
Warm-Up: The DongChenWen13 Approach

\( s_1, \ldots, s_m \)

\( m \) out of \( mm \) secret sharing of \( ss \)

\( 0, 1 \leq 0, 1, 0, 1, 0, 1, 0, 1 \leq \kappa \)

\( 1, 2, 1, 2, mm \) ones

Goal — restrict the Receiver to a valid Bloom filter

- Bloom filter of \( m \) bits contains \( 1, 2, m \) ones

- Make Receiver prove zero choice bits
  - \( s_1, 1, 1, \ldots, s_m \)
  - Sample random key \( s \leftarrow \{0,1\}^\kappa \)
  - Generate a \( \frac{m}{2} \) out of \( m \) secret sharing of \( s \)
    - \( s_1, \ldots, s_m \)

- Transmit \( s_i \) as the \( i \)th zero OT message
- Encrypt summary values with \( s \)
Warm-Up: The DongChenWen13 Approach

as the \(i\)th zero OT message

\(s_1, \ldots, s_m\) secret sharing of \(s\)

\(m\) 2 out of \(mm\) bits contains \(1\) \(2m\) ones

Goal — restrict the Receiver to a valid Bloom filter

* Bloom filter of \(m\) bits contains 1 \(2m\) ones
* Make Receiver prove zero choice bits
  * \(s_1, \ldots, s_m\)
  * Transmit \(s_i\) as the \(i\)th zero OT message
  * Generate a \(\frac{m}{2}\) out of \(m\) secret sharing of \(s\)
  * \(s_1, \ldots, s_m\)
  * Transmit \(s_i\) as the \(i\)th zero OT message
  * Encrypt summary values with \(s\)
Warm-Up: The DongChenWen13 Approach

as the $i$th zero OT message
$s_1,..., s_m s s s m m m s m$
m $2$ out of $mm$ secret sharing of $ss$
$0,1 \kappa 0,1 0,1 0,1 \kappa \kappa \kappa 0,1 \kappa$

Goal — restrict the Receiver to a valid Bloom filter
- Bloom filter of $m$ bits contains 1 $2 m$ ones
- Make Receiver prove zero choice bits
  - $s_1 1 1, ..., s_m$
  - Encrypt summary values with $s$ Generate a $\frac{m}{2}$ out of $m$ secret sharing of $s$
    - $s_1, ..., s_m$
  - Transmit $s_i$ as the $i$th zero OT message
  - Encrypt summary values with $s$

$$X = \{a, b\}$$

$$Y = \{a, c\}$$

- Transmit $s_i$ as the $i$th zero OT message
  - Encrypt summary values with $s$
- Make Receiver prove zero choice bits
  - $s_1 1 1, ..., s_m$
  - Encrypt summary values with $s$
Warm-Up: The DongChenWen13 Approach

as the $i$th zero OT message

$s \ 1, \ldots, \ s \ m \ s \ s \ s \ m \ m \ m \ s \ m$

$m \ 2$ out of $mm$ secret sharing of $ss$

$0,1 \ \kappa \ 0,1 \ 0,1 \ 0,1 \ \kappa \ \kappa \ 0,1 \ \kappa$

1 2 2 1 2 $mm$ ones

Goal — restrict the Receiver to a valid Bloom filter

• Bloom filter of $m$ bits contains 1 2 $m$ ones

• Make Receiver prove zero choice bits

  • $s \ 1 \ 1 , \ \ldots, \ s \ m$

  • Encrypt summary values with $s$

    Generate a $\frac{m}{2}$ out of $m$ secret sharing of $s$

    • $s_1, \ldots, s_m$

• Transmit $s_i$ as the $i$th zero OT message

• Encrypt summary values with $s$

$X = \{a, b\}$

$Y = \{a, c\}$

\[
\begin{array}{c|c}
  s_0 & m_0 \\
  s_1 & m_1 \\
  s_2 & m_2 \\
  s_3 & m_3 \\
  s_4 & m_4 \\
  s_5 & m_5 \\
  s_6 & m_6 \\
\end{array}
\]

\[
X = \{m_0 \oplus m_5, m_2 \oplus m_3\}
\]

\[
Y = \{m_0 \oplus m_5, m_3 \oplus m_6\}
\]

Output:

$\mathbb{D}_s (X) \cap \{m_0 \oplus m_5, m_3 \oplus m_6\}$
as the $i$th zero OT message

$s_1, ..., s_m s s s m m m s m$

$m$ 2 out of $mm$ secret sharing of $s s$

$0,1 \kappa 0,1 0,1 0,1 \kappa \kappa 0,1 \kappa$

Goal — restrict the Receiver to a valid Bloom filter

• Bloom filter of $m$ bits contains 12 $m$ ones

• Make Receiver prove zero choice bits
  
  • $s_1 1 1, ..., s_m$
  
  • Encrypt summary values with $s$

• Transmit $s_i$ as the $i$th zero OT message

• Encrypt summary values with $s$

\[
\begin{align*}
X &= \{a, b\} \\
Y &= \{a, c\}
\end{align*}
\]

\[
\begin{array}{c|c}
  & m_0 \\
\hline
s_0 & m_0 \\
s_1 & m_1 \\
s_2 & m_2 \\
s_3 & m_3 \\
s_4 & m_4 \\
s_5 & m_5 \\
s_6 & m_6
\end{array}
\]

\[
\begin{align*}
X = \{a, b\} & \quad \text{Output:} \\
Y = \{a, c\} & \quad \mathbb{D}_s (X) \cap \{m_0 \oplus m_5, m_3 \oplus m_6\}
\end{align*}
\]
Warm-Up: The DongChenWen13 Approach

- **Goal** — restrict the Receiver to a valid Bloom filter
  - Bloom filter of $m$ bits contains $\frac{1}{2}m$ ones
- **Make Receiver prove zero choice bits**
  - Sample random key $s \leftarrow \{0,1\}^\kappa$
  - Generate a $\frac{m}{2}$ out of $m$ secret sharing of $s$
    - $s_1, ..., s_m$
  - Transmit $s_i$ as the $i$th zero OT message
  - Encrypt summary values with $s$

**Table:**

<table>
<thead>
<tr>
<th>$s_0$</th>
<th>$m_0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_1$</td>
<td>$m_1$</td>
</tr>
<tr>
<td>$s_2$</td>
<td>$m_2$</td>
</tr>
<tr>
<td>$s_3$</td>
<td>$m_3$</td>
</tr>
<tr>
<td>$s_4$</td>
<td>$m_4$</td>
</tr>
<tr>
<td>$s_5$</td>
<td>$m_5$</td>
</tr>
<tr>
<td>$s_6$</td>
<td>$m_6$</td>
</tr>
</tbody>
</table>

**Output:**

$X = \{a, b\}$

$Y = \{a, c\}$
Is this secure?

- \( \frac{m}{2} \) ones
- Selective failure attack by the Sender...

\[
Y = \{a, c\}
\]

\[
X = \{a, b\}
\]

\[
\begin{array}{cccc}
  s_0 & m_0 \\
  s_1 & m_1 \\
  s_2 & m_2 \\
  s_3 & m_3 \\
  s_4 & m_4 \\
  s_5 & m_5 \\
  s_6 & m_6 \\
\end{array}
\]

\[
\begin{array}{cccc}
  m_0 & 1 \\
  s_1 & 0 \\
  s_2 & 0 \\
  s_3 & 1 \\
  s_4 & 0 \\
  m_5 & 1 \\
  m_6 & 1 \\
\end{array}
\]

Output:

\[
\bar{X} = E_s \left( \left\{ m_0 \oplus m_5, m_2 \oplus m_3 \right\} \right) \quad D_s(\bar{X}) \cap \left\{ m_0 \oplus m_5, m_3 \oplus m_6 \right\}
\]
Warm-Up: The DongChenWen13 Approach

$m \leq \frac{m}{2}$ ones

Is this secure?

- Receiver is forced to use $\leq \frac{m}{2}$ ones
- Selective failure attack by the Sender...

\[
X = \{a, b\}
\]
\[
Y = \{a, c\}
\]

Output:

\[
\hat{X} = \mathbb{E}_s \left( \left\{ m_0 \oplus m_5, m_2 \oplus m_3 \right\} \right) \cap \mathbb{D}_s (\hat{X}) \cap \left\{ m_0 \oplus m_5, m_3 \oplus m_6 \right\}
\]
Warm-Up: The DongChenWen13 Approach

$m \ 2 \ mm \ m \ 2 \ 2 \ m \ 2 \ ones$

Is this secure?

• Selective failure attack by the Sender…
  Selective failure attack by the Sender…

\[
\begin{array}{c|c}
    s_0 & m_0 \\
    s_1 & m_1 \\
    s_2 & m_2 \\
    s_3 & m_3 \\
    s_4 & m_4 \\
    s_5 & m_5 \\
    s_6 & m_6 \\
\end{array}
\]

\[
\begin{array}{c|c}
    & m_0 \\
    s_0 & 0 \\
    s_1 & 0 \\
    s_2 & 1 \\
    s_3 & 0 \\
    s_4 & 1 \\
    m_5 & 1 \\
    m_6 & 1 \\
\end{array}
\]

Output:

\[
\mathcal{X} = \mathbb{E}_s \left( \left\{ m_0 \oplus m_5, m_2 \oplus m_3 \right\} \right) \rightarrow \mathcal{D}_s (\mathcal{X}) \cap \left\{ m_0 \oplus m_5, m_3 \oplus m_6 \right\}
\]

[Rindal Rosulek 17, Lambaek 17]
Warm-Up: The DongChenWen13 Approach

• Is this secure?
  • Receiver is forced to use $\leq \frac{m}{2}$ ones
  • Selective failure attack by the Sender…

• Example Attack:
  • replace $s_4$ with random value $r$

$Y = \{a, c\}$

Output:

$\hat{X} = \mathbb{E}_s (\{m_0 \oplus m_5, m_2 \oplus m_3\}) \xrightarrow{\mathcal{D}_s (\hat{X})} \{m_0 \oplus m_5, m_0 \oplus m_5, m_3 \oplus m_6\}$

[RindalRosulek17, Lambaek17]
Warm-Up: The DongChenWen13 Approach

Is this secure?

• Receiver is forced to use $\leq \frac{m}{2}$ ones
• Selective failure attack by the Sender...

• Example Attack:
  • replace $s_4$ with random value $r$
  • Can not reconstruct $s$ if $r$ is picked up

• $\forall y \in Y : h_i(y) \neq 4$
• Can not be simulated!

$X = \{a, b\}$

$Y = \{a, c\}$

Output:

$X = \mathbb{E}_s \left( \{m_0 \oplus m_5, m_2 \oplus m_3\} \right)$

$\mathbb{D}_s(X) \cap \{m_0 \oplus m_5, m_3 \oplus m_6\}$

[RindalRosulek17, Lambaek17]
Warm-Up: The DongChenWen13 Approach

∀y ∈ Y : h_i h_i i h i y y y y ≠ 4

Is this secure?

- Receiver is forced to use \( \leq \frac{m}{2} \) ones
- Selective failure attack by the Sender...

- Example Attack:
  - replace \( s_4 \) with random value \( r \)
  - Can not reconstruct \( s \) if \( r \) is picked up
  - Can not be simulated! Can not be simulated!
Cut and Choose Approach

Make Receiver $p$

<table>
<thead>
<tr>
<th></th>
<th>$r_0$</th>
<th>$m_0$</th>
<th>$r_1$</th>
<th>$m_1$</th>
<th>$r_2$</th>
<th>$m_2$</th>
<th>$r_3$</th>
<th>$m_3$</th>
<th>$r_4$</th>
<th>$m_4$</th>
<th>$r_5$</th>
<th>$m_5$</th>
<th>$r_6$</th>
<th>$m_6$</th>
<th>$r_7$</th>
<th>$m_7$</th>
<th>$r_8$</th>
<th>$m_8$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

[Random]

[1] Rindal Rosulek 17
Cut and Choose Approach

Make Receiver p

• Sender challenges on a subset of OT
  • Receiver must reveal select bits

\[ RindalRosulek17 \]
Cut and Choose Approach

Make Receiver p

• Sender challenges on a subset of OT
  • Receiver must reveal select bits

[Ref: RindalRosulek17]
Cut and Choose Approach

- Make Receiver \( p = 2 \) zero bits, aborts otherwise
  - Sender challenges on a subset of OT
    - Receiver must reveal select bits
    - Expect to see 12 zero bits, aborts otherwise

\[
\begin{array}{cc}
r_0 & m_0 \\
r_1 & m_1 \\
r_2 & m_2 \\
r_3 & m_3 \\
r_4 & m_4 \\
r_5 & m_5 \\
r_6 & m_6 \\
r_7 & m_7 \\
r_8 & m_8 \\
\end{array}
\]

\[
\begin{array}{cc}
r_0 & 0 \\
r_1 & 1 \\
r_2 & 0 \\
m_3 & 1 \\
m_4 & 1 \\
m_5 & 0 \\
m_6 & 1 \\
m_7 & 0 \\
m_8 & 1 \\
\end{array}
\]

[RindalRosulek17]
Cut and Choose Approach

- Make Receiver prove zero bits in an input-independent way
- Receiver uses \textit{random} OT select bits
- Sender challenges on a subset of OT
  - Receiver must reveal select bits
  - Expect to see $\frac{1}{2}$ zero bits, aborts otherwise

\[ \begin{array}{c|c} r_0 & m_0 \\ r_1 & m_1 \\ r_3 & m_3 \\ r_4 & m_4 \\ r_5 & m_5 \\ r_7 & m_7 \\ r_8 & m_8 \\ \end{array} \quad \begin{array}{c|c} r_0 & 0 \\ m_1 & 1 \\ m_3 & 1 \\ m_4 & 1 \\ m_5 & 0 \\ m_7 & 0 \\ m_8 & 1 \\ \end{array} \]
Cut and Choose Approach

- Issue: Remaining OTs do not form valid Bloom filter

\[ Y = \{a, c\} \]

\[
\begin{array}{c|c}
 r_0 & m_0 \\
 r_1 & m_1 \\
 r_3 & m_3 \\
 r_4 & m_4 \\
 r_5 & m_5 \\
 r_7 & m_7 \\
 r_8 & m_8 \\
\end{array}
\]

\[
\begin{array}{c|c}
 r_0 & m_0 \\
 r_1 & m_1 \\
 r_3 & m_3 \\
 r_4 & m_4 \\
 r_5 & m_5 \\
 r_7 & m_7 \\
 r_8 & m_8 \\
\end{array}
\]

\[
\begin{array}{c|c}
 r_0 & 0 \\
 r_1 & 1 \\
 r_3 & 1 \\
 r_4 & 1 \\
 r_5 & 0 \\
 r_7 & 0 \\
 r_8 & 1 \\
\end{array}
\]

\( h_i(a) \)

\( h_i(c) \)
Cut and Choose Approach

\[ Y = \{a, c\} \]

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>( r_0 )</td>
<td>( m_0 )</td>
<td>( r_1 )</td>
<td>( m_1 )</td>
</tr>
<tr>
<td>( r_3 )</td>
<td>( m_3 )</td>
<td>( r_4 )</td>
<td>( m_4 )</td>
</tr>
<tr>
<td>( r_5 )</td>
<td>( m_5 )</td>
<td>( r_7 )</td>
<td>( m_7 )</td>
</tr>
<tr>
<td>( r_8 )</td>
<td>( m_8 )</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[ r_0 \overline{m_0} \]

\[ r_1 \overline{m_1} \]

\[ r_3 \overline{m_3} \]

\[ r_4 \overline{m_4} \]

\[ r_5 \overline{m_5} \]

\[ r_7 \overline{m_7} \]

\[ r_8 \overline{m_8} \]

\[ h_i(a) \]

\[ h_i(c) \]

[Reference: Rindal and Rosulek 2017]
Cut and Choose Approach

Random OTs → desired BF

- Randomly permute OTs to form Bloom filter
  - $\pi$ random OTs → desired BF
Cut and Choose Approach

- Issue: Remaining OTs do not form valid Bloom filter
- Constructs desired Bloom filter
- Randomly permute OTs to form Bloom filter
  - $\pi$ (random OTs) $\rightarrow$ desired BF

\[ Y = \{a, c\} \]

\[
\begin{array}{c|c|c}
| r_4 | m_4 | \hline
| r_1 | m_1 | 1 \hline
| r_3 | m_3 | 1 \hline
| r_0 | m_0 | 0 \hline
| r_5 | m_5 | 0 \hline
| r_7 | m_7 | 0 \hline
| r_8 | m_8 | 1 \hline
\end{array}
\]

\[
\begin{array}{c|c|c}
| m_4 | \hline
| m_1 | 1 \hline
| m_3 | 0 \hline
| r_0 | 0 \hline
| r_5 | 0 \hline
| r_7 | 1 \hline
| m_8 | 1 \hline
\end{array}
\]

$\pi$
Cut and Choose Approach

- **Issue**: Remaining OTs do not form valid Bloom filter
- **Constructs desired Bloom filter**
- **Randomly permute OTs to form Bloom filter**
  - \( \pi(\text{random } OTs) \rightarrow \text{desired } BF \)

\[
Y = \{a, c\}
\]

\[
\begin{array}{c|c|c|c|c}
0 & 1 & 0 & 1 & 1 \\
1 & 0 & 0 & 1 & 1 \\
0 & 0 & 1 & 1 & 1 \\
0 & 1 & 1 & 1 & 1 \\
\end{array}
\]

\( \pi \)
Cut and Choose Approach

- **Issue**: Remaining OTs do not form valid Bloom filter
- **Constructs desired Bloom filter**
- **Randomly permute OTs to form Bloom filter**
  - $\pi$(random OTs) $\to$ desired $BF$

- $Y = \{a, c\}$

\[
\begin{array}{c c c}
  r_4 & m_4 & r_0 \\
  r_0 & m_0 & r_7 \\
  r_7 & m_7 & r_5 \\
  r_5 & m_5 & r_3 \\
  r_3 & m_3 & r_8 \\
  r_8 & m_8 & r_0 \\
\end{array}
\]

\[
\begin{array}{c c c}
  m_4 & 1 & 1 \\
  r_0 & 0 & 0 \\
  r_7 & 0 & 0 \\
  m_5 & 1 & 1 \\
  r_5 & 0 & 0 \\
  m_3 & 0 & 0 \\
  m_8 & 1 & 1 \\
\end{array}
\]

$h_i(a)$ and $h_i(c)$
Cut and Choose Approach

• Issue: Remaining OTs do not form valid Bloom filter
• Constructs desired Bloom filter
• Randomly permute OTs to form Bloom filter
  • $\pi$(random OTs) $\rightarrow$ desired BF

$Y = \{a, c\}$

\[
\begin{array}{cc}
\begin{array}{cc}
r_4 & m_4 \\
r_0 & m_0 \\
r_7 & m_7 \\
r_8 & m_8 \\
r_5 & m_5 \\
r_3 & m_3 \\
r_1 & m_1 \\
\end{array} & \begin{array}{cc}
m_4 & 1 \\
m_0 & 0 \\
m_7 & 0 \\
m_8 & 1 \\
m_5 & 1 \\
m_3 & 0 \\
m_1 & 1 \\
\end{array}
\end{array}
\]

$h_i(a)$

$h_i(c)$
Cut and Choose Approach

• Issue: Remaining OTs do not form valid Bloom filter
• Constructs desired Bloom filter
• Randomly permute OTs to form Bloom filter
  • $\pi$(random OTs) → desired BF

$\begin{align*}
Y &= \{a, c\} \\
\begin{array}{c|c}
\hline
r_4 & m_4 \\
r_0 & m_0 \\
r_7 & m_7 \\
r_8 & m_8 \\
r_5 & m_5 \\
r_3 & m_3 \\
r_1 & m_1 \\
\hline
\end{array}
\end{align*}$
Cut and Choose Approach

- Issue: Remaining OTs do not form valid Bloom filter
- Constructs desired Bloom filter
- Randomly permute OTs to form Bloom filter
  - $\pi$(random OTs) → desired BF

[RindalRosulek17]
Cut and Choose Approach

- Issue: Remaining OTs do not form valid Bloom filter
- Constructs desired Bloom filter
- Randomly permute OTs to form Bloom filter
  - $\pi(\text{random} \ OTs) \rightarrow \text{desired} \ BF$

$Y = \{a, c\}$

$$Y = \{a, c\}$$

$$h_i(a)$$

$$h_i(c)$$

$\pi$
Cut and Choose Approach

- Issue: Remaining OTs do not form valid Bloom filter
- Constructs desired Bloom filter
- Randomly permute OTs to form Bloom filter
  - $\pi$ (random OTs) → desired BF

Output:

$$\hat{X} = \{m_4 \oplus m_1, m_2 \oplus m_3\}$$

$$\hat{X} \cap \{m_4 \oplus m_1, m_8 \oplus m_3\}$$

$Y = \{a, c\}$

\[ Y = \{a, c\} \]

[ RindalRosulek17 ]
Cut and Choose Parameters

- Issue: Random OTs/Cut-and-Choose may not result in exactly \( \frac{1}{2} \) zero select bits!
Cut and Choose Parameters

- Issue: Random OTs/Cut-and-Choose may not result in exactly $\frac{1}{2}$ zero select bits!
Cut and Choose Parameters

- Issue: Random OTs/Cut-and-Choose may not result in exactly $\frac{1}{2}$ zero select bits!

[Reference: Rindal Rosulek 17]

\[
\begin{array}{cccc}
\text{Random} & 0 & 1 & 0 \\
1 & m_1 & m_1 & r_0 \\
1 & m_6 & m_6 & r_6 \\
1 & m_7 & m_7 & r_7 \\
1 & m_8 & m_8 & r_8 \\
\end{array}
\]
Cut and Choose Parameters

- Issue: Random OTs/Cut-and-Choose may not result in exactly $\frac{1}{2}$ zero select bits!
- Need robust checking of zero bits
Cut and Choose Parameters

Issue: Random OTs/Cut-and-Choose may not result in exactly $\frac{1}{2}$ zero select bits!

- Need robust checking of zero bits
- Desired properties:
  - $\Pr[\text{good guy accused}] \leq \text{neg}(\kappa)$
- Sufficient to check 1% of the OTs!
Cut and Choose Parameters

Issue: Random OTs/Cut-and-Choose may not result in exactly $\frac{1}{2}$ zero select bits!

- Need robust checking of zero bits
- Desired properties:
  - $\Pr[\text{good guy accused}] \leq \text{neg}(\kappa)$
  - 
- Sufficient to check 1% of the OTs!
Cut and Choose Parameters

Issue: Random OTs/Cut-and-Choose may not result in exactly $\frac{1}{2}$ zero select bits!

• Need robust checking of zero bits

• Desired properties:
  • Pr[good guy accused] ≤ neg($\kappa$)

• Sufficient to check 1% of the OTs!

$E[\text{good guy}]$
Cut and Choose Parameters

Issue: Random OTs/Cut-and-Choose may not result in exactly $\frac{1}{2}$ zero select bits!

- Need robust checking of zero bits
- Desired properties:
  - $\Pr[good\ guy\ accused] \leq neg(\kappa)$
- Use Chernoff Bounds
- Sufficient to check 1% of the OTs!
Cut and Choose Parameters

\[ \Pr \text{ Bad guy not caught} \leq \text{neg}(\kappa) \]
\[ \text{Bad guy not caught} \leq \text{neg}(\kappa) \]

\[ \Pr \text{ Bad guy not caught} \leq \text{neg}(\kappa) \]

**Issue:** Random OTs/Cut-and-Choose may not result in exactly $\frac{1}{2}$ zero select bits!

- Need robust checking of zero bits
- Desired properties:
  - $\Pr[\text{good guy accused}] \leq \text{neg}(\kappa)$
  - $\Pr \text{ Bad guy not caught} \leq \text{neg}(\kappa)$
- Use Chernoff Bounds
- Sufficient to check 1% of the OTs!
Cut and Choose Parameters

\[ \text{Pr Bad guy not caught} \leq \text{neg}(\kappa) \]
\[ \text{Bad guy not caught} \leq \text{neg}(\kappa) \]

**Issue:** Random OTs/Cut-and-Choose may not result in exactly \( \frac{1}{2} \) zero select bits!

- Need robust checking of zero bits
- Desired properties:
  - \( \text{Pr[good guy accused]} \leq \text{neg}(\kappa) \)
  - \( \text{Pr Bad guy not caught} \leq \text{neg}(\kappa) \)

- Use Chernoff Bounds
- Sufficient to check 1% of the OTs!
Cut and Choose Parameters

\( r \Pr \text{ Bad guy not caught } \leq \text{neg}(\kappa) \) Bad guy not caught BBa
add gguyuy mnoott ccawuggh tt Bad guy not caught \leq \text{ne}
egg(\kappa) Pr Bad guy not caught \leq \text{neg}(\kappa)

*Issue:* Random OTs/Cut-and-Choose may not
result in exactly \( \frac{1}{2} \) zero select bits!

- Need robust checking of zero bits
- Desired properties:
  - \( \Pr[\text{good guy accused}] \leq \text{neg}(\kappa) \)
  - \( \Pr \text{ Bad guy not caught } \leq \text{neg}(\kappa) \)
- Use Chernoff Bounds
- Sufficient to check 1% of the OTs!

\[ t \ll E[\text{good guy}] \]

Abort threshold

\#zeros seen
Cut and Choose Parameters

Issue: Random OT/Cut-and-Choose may not result in exactly $\frac{1}{2}$ zero select bits!

• Need robust checking of zero bits
• Desired properties:
  • $\text{Pr}[\text{good guy accused}] \leq \text{neg}(\kappa)$
  • $\text{Pr} \text{ Bad guy not caught} \leq \text{neg}(\kappa)$

• Use Chernoff Bounds
• Sufficient to check 1% of the OTs!
Cut and Choose Parameters

\% of the OTs!

\[ r \Pr \text{ Bad guy not caught } \leq \text{neg}(\kappa) \]

\[ \Pr \text{ Bad guy not caught } \leq \text{neg}(\kappa) \]

\[ \Pr [\text{good guy accused}] \leq \text{neg}(\kappa) \]

\[ \Pr [\text{Bad guy not caught}] \leq \text{neg}(\kappa) \]

\[ \frac{1}{2} \text{ zero select bits!} \]

\[ \Pr \text{ Bad guy not caught } \leq \text{neg}(\kappa) \]

Issue: Random OTs/Cut-and-Choose may not result in exactly \( \frac{1}{2} \) zero select bits!

- Need robust checking of zero bits
- Desired properties:
  - \( \Pr [\text{good guy accused}] \leq \text{neg}(\kappa) \)
  - \( \Pr \text{ Bad guy not caught } \leq \text{neg}(\kappa) \)

\[ \Pr \text{ Bad guy not caught } \leq \text{neg}(\kappa) \]

- Sufficient to check 1\% of the OTs!
- Sufficient to check 1\% of the OTs!
Cut and Choose Parameters

% of the OTs!
% of the OTs!

\[ r \Pr \text{ Bad guy not caught} \leq \text{neg}(\kappa) \]
\[ \Pr \text{ Bad guy not caught} \leq \text{neg}(\kappa) \]

**Issue:** Random OTs/Cut-and-Choose may not result in exactly \( \frac{1}{2} \) zero select bits!

- Need robust checking of zero bits
- Desired properties:
  - \( \Pr[\text{good guy accused}] \leq \text{neg}(\kappa) \)
  - \( \Pr \text{ Bad guy not caught} \leq \text{neg}(\kappa) \)
- Sufficient to check 1% of the OTs!
- Sufficient to check 1% of the OTs!
- Sufficient to check 1% of the OTs!
Extracting $Y$ with Random Oracle

Simulator must extract the effective input $Y$

- $F$ is not naturally invertible
- $BF$ may be malformed…

- Solution:
  - Model hash function $h_i(\cdot)$ as Random Oracle
  - Non-programmable RO

\[
Y = \{a, c\}
\]

\[
\begin{array}{c|c}
  r_4 & m_4 \\
r_0 & m_0 \\
r_7 & m_7 \\
r_8 & m_8 \\
r_5 & m_5 \\
r_1 & m_1 \\
r_3 & m_3 \\
\end{array}
\]

\[
\begin{array}{c|c}
m_4 & 1 \\
r_0 & 0 \\
r_7 & 0 \\
r_8 & 1 \\
r_5 & 0 \\
m_1 & 1 \\
m_3 & 1 \\
\end{array}
\]

Output:

\[
\hat{X} = \{m_4 \oplus m_1, m_2 \oplus m_3\}
\]

\[
\hat{X} \cap \{m_4 \oplus m_1, m_8 \oplus m_3\}
\]

[RindalRosulek17]
Extracting $Y$ with Random Oracle

Simulator must extract the effective input $Y$

- $F$ is not naturally invertible
- $BF$ may be malformed…

Solution:
- Model hash function $h_i(\cdot)$ as Random Oracle
- Non-programmable RO

$X \cap Y = \{a\}$

$Y = \{a, c\}$

$OT\quad OT\quad OT\quad h_i(a)$

$OT\quad OT\quad h_i(c)$
Extracting $Y$ with Random Oracle

Simulator must extract the effective input $Y$

- $F$ is not naturally invertible
- $BF$ may be malformed…

Solution:
- Model hash function $h_i(\cdot)$ as Random Oracle
- Non-programmable RO
Extracting $Y$ with Random Oracle

Simulator must extract the effective input $Y$

- Can extract OT select bits

- $F$ is not naturally invertible
- $BF$ may be malformed…

Solution:
- Model hash function $h_i(\cdot)$ as Random Oracle
- Non-programmable RO

$Y = \{a, c\}$
Extracting $Y$ with Random Oracle

is not naturally invertible

Simulator must extract the effective input $Y$

- Can extract OT select bits

- Issues:
  - $BF$ is not naturally invertible
  - $BF$ may be malformed...

- Solution:
  - Model hash function $h_i(\cdot)$ as Random Oracle
  - Non-programmable RO

\[
\begin{align*}
Y &= \{a, c\} \\
BF &= \begin{bmatrix}
1 \\
0 \\
0 \\
1 \\
0 \\
1 \\
1
\end{bmatrix} \\
\text{Simulator}
\end{align*}
\]
Extracting $Y$ with Random Oracle

may be malformed…
is not naturally invertible
Simulator must extract the effective input $Y$
  • Can extract OT select bits

• Issues:
  • $BF$ may be malformed…
  • $BF$ may be malformed…

• Solution:
  • Model hash function $h_i(\cdot)$ as Random Oracle
  • Non-programmable RO

[1] Rindal Rosulek17
Extracting $Y$ with Random Oracle

$h_i \mathbin{\mathsf{hash}} (\cdot)$ as Random Oracle

may be malformed...

is not naturally invertible

Simulator must extract the effective input $Y$

• Can extract OT select bits

• Issues:
  • $BF$ may be malformed...

• Solution:
  • Model hash function $h_i (\cdot)$ as Random Oracle

• Solution:
  • Model hash function $h_i (\cdot)$ as Random Oracle
  • Non-programmable RO

\[
\text{Simulator} \quad BF = \begin{bmatrix}
1 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 1 & 0 \\
1 & 0 & 0 & 1 & 1 \\
\end{bmatrix} \quad \text{OT} \quad Y = \{a, c\}
\]

$[\text{RindalRosulek17}]$
Extracting $Y$ with Random Oracle

$h_i \cdot i \cdot h_i(\cdot)$ as Random Oracle

may be malformed…

is not naturally invertible

Simulator must extract the effective input $Y$
  • Can extract OT select bits

• Issues:
  • $BF$ may be malformed…

• Solution:
  • Model hash function $h_i(\cdot)$ as Random Oracle

• Solution:
  • Model hash function $h_i(\cdot)$ as Random Oracle
  • Non-programmable RO
Extracting $Y$ with Random Oracle

$h_i ii h_i (\cdot)$ as Random Oracle

may be malformed…

is not naturally invertible

Simulator must extract the effective input $Y$
  - Can extract OT select bits

• Issues:
  - $BF$ may be malformed…

• Solution:
  - Model hash function $h_i (\cdot)$ as Random Oracle

• Solution:
  - Model hash function $h_i (\cdot)$ as Random Oracle
  - Non-programmable RO

$Y = \{a, c\}$
Extracting $Y$ with Random Oracle

$h_i(i) \cdot$ as Random Oracle
may be malformed...
is not naturally invertible
Simulator must extract the effective input $Y$
• Can extract OT select bits

• Issues:
  • $BF$ may be malformed...
• Solution:
  • Model hash function $h_i(\cdot)$ as Random Oracle

• Solution:
  • Model hash function $h_i(\cdot)$ as Random Oracle
  • Non-programmable RO
Extracting $Y$ with Random Oracle

$h_i \circ h_i(\cdot)$ as Random Oracle may be malformed…

is not naturally invertible

Simulator must extract the effective input $Y$

- Can extract OT select bits

- Issues:
  - $BF$ may be malformed…
  - Solution:
    - Model hash function $h_i(\cdot)$ as Random Oracle
    - Solution:
      - Model hash function $h_i(\cdot)$ as Random Oracle
      - Non-programmable RO
Extracting $Y$ with Random Oracle

$h_i \circ h_i (\cdot)$ as Random Oracle

may be malformed...

is not naturally invertible

Simulator must extract the effective input $Y$
  - Can extract OT select bits

• Issues:
  - $BF$ may be malformed...
• Solution:
  - Model hash function $h_i (\cdot)$ as Random Oracle

• Solution:
  - Model hash function $h_i (\cdot)$ as Random Oracle
  - Non-programmable RO
Extracting $Y$ with Random Oracle

$h_ii h_i(\cdot)$ as Random Oracle

may be malformed…

is not naturally invertible

Simulator must extract the effective input $Y$

- Can extract OT select bits

Issues:

- $BF$ may be malformed…

Solution:

- Non-programmable ROSolution:
- Model hash function $h_i(\cdot)$ as Random Oracle
- Non-programmable RO
Bloom filter of size $\sim 2nk$ allows a Receiver to insert $n$ items

$$F(a) = m_4 \oplus m_1$$
$$F(c) = m_8 \oplus m_3$$

- $F(\cdot)$
- View Bloom filter protocol as an OPRF

$Y = \{a, c\}$

$$Y = \{a, c\}$$

$$\begin{array}{cc}
r_4 & m_4 \\
r_0 & m_0 \\
r_7 & m_7 \\
r_8 & m_8 \\
r_5 & m_5 \\
r_1 & m_1 \\
r_3 & m_3 \\
\end{array}$$

$$\begin{array}{cccc}
m_4 & 1 \\
r_0 & 0 & r_0 \\
m_7 & 0 & r_7 \\
m_8 & 1 & r_8 \\
m_5 & 0 & r_5 \\
m_1 & 1 & r_1 \\
m_3 & 1 & r_3 \\
\end{array}$$

$$\{m_4 \oplus m_1, m_8 \oplus m_3\}$$
Generalized Encodings

\[ F \cdot \]

Bloom filter of size \( \sim 2n\kappa \) allows a Receiver to insert \( n \) items

\[ F(a) = m_4 \oplus m_1 \]
\[ F(c) = m_8 \oplus m_3 \]

- Sender can generate any encoding \( F(\cdot) \)
- View Bloom filter protocol as an OPRF

\[ Y = \{a, c\} \]

\[
\begin{array}{c|c}
  r_4 & m_4 \\
  r_0 & m_0 \\
  r_7 & m_7 \\
  r_8 & m_8 \\
  r_5 & m_5 \\
  r_1 & m_1 \\
  r_3 & m_3 \\
\end{array}
\]

\[
\begin{array}{c|c}
  m_4 & 1 \\
  r_0 & 0 \\
  r_7 & 0 \\
  m_8 & 1 \\
  r_5 & 0 \\
  m_1 & 1 \\
  m_3 & 1 \\
\end{array}
\]

\[ \{m_4 \oplus m_1, m_8 \oplus m_3\} \]
Generalized Encodings

\( FFF(\cdot) \)

Bloom filter of size \( \sim 2n\kappa \) allows a Receiver to insert \( n \) items

\[
F(a) = m_4 \oplus m_1 \\
F(c) = m_8 \oplus m_3
\]

- View Bloom filter protocol as an OPRF
- View Bloom filter protocol as an OPRF

\[ Y = \{a, c\} \]

\[
\begin{array}{c|c}
  r_4 & m_4 \\
  r_0 & m_0 \\
  r_7 & m_7 \\
  r_8 & m_8 \\
  r_5 & m_5 \\
  r_1 & m_1 \\
  r_3 & m_3 \\
\end{array}
\]

\[
\begin{array}{c|c}
  m_4 & 1 \\
  r_0 & 0 \\
  r_7 & 0 \\
  m_8 & 1 \\
  r_5 & 0 \\
  m_1 & 1 \\
  m_3 & 1 \\
\end{array}
\]

\( h_i(a) \)

\( h_i(c) \)
Generalized Encodings

\[ F \neq F(\cdot) \]

Bloom filter of size \( \sim 2n\kappa \) allows a Receiver to insert \( n \) items

\[ F(a) = m_4 \oplus m_1 \]
\[ F(c) = m_8 \oplus m_3 \]

- View Bloom filter protocol as an OPRF
- View Bloom filter protocol as an OPRF

\[ Y = \{a, c\} \]

\[
\begin{array}{c|c}
 r_4 & m_4 \\
r_0 & m_0 \\
r_7 & m_7 \\
r_8 & m_8 \\
r_5 & m_5 \\
r_1 & m_1 \\
r_3 & m_3 \\
\end{array}
\]

\[ m_4 \oplus m_1, \quad m_8 \oplus m_3 \]
Comparison – De Cristofaro, Kim, Tsudik10

• DKT10 - Malicious Diffie-Hellman style approach: $x^{\alpha \beta} = y^{\beta \alpha}$
Comparison – De Cristofaro, Kim, Tsudik10

- DKT10 - Malicious Diffie-Hellman style approach: \( x^{\alpha \beta} = y^{\beta \alpha} \)
Comparison – De Cristofaro, Kim, Tsudik10

- DKT10 - Malicious Diffie-Hellman style approach: $x^{\alpha \beta} = y^{\beta \alpha}$
Comparison – De Cristofaro, Kim, Tsudik 10

• DKT10 - Malicious Diffie-Hellman style approach: \( x^\alpha \beta = y^\beta \alpha \)

![Graph showing running time and communication (MB) for different protocols DCW13, DKT10, RR17.](image-url)
Comparison – De Cristofaro, Kim, Tsudik10

- DKT10 - Malicious Diffie-Hellman style approach: \( x^{\alpha \beta} = y^{\beta \alpha} \)
Comparison – De Cristofaro, Kim, Tsudik10

- DKT10 - Malicious Diffie-Hellman style approach: $x^{\alpha \beta} = y^{\beta \alpha}$
The End