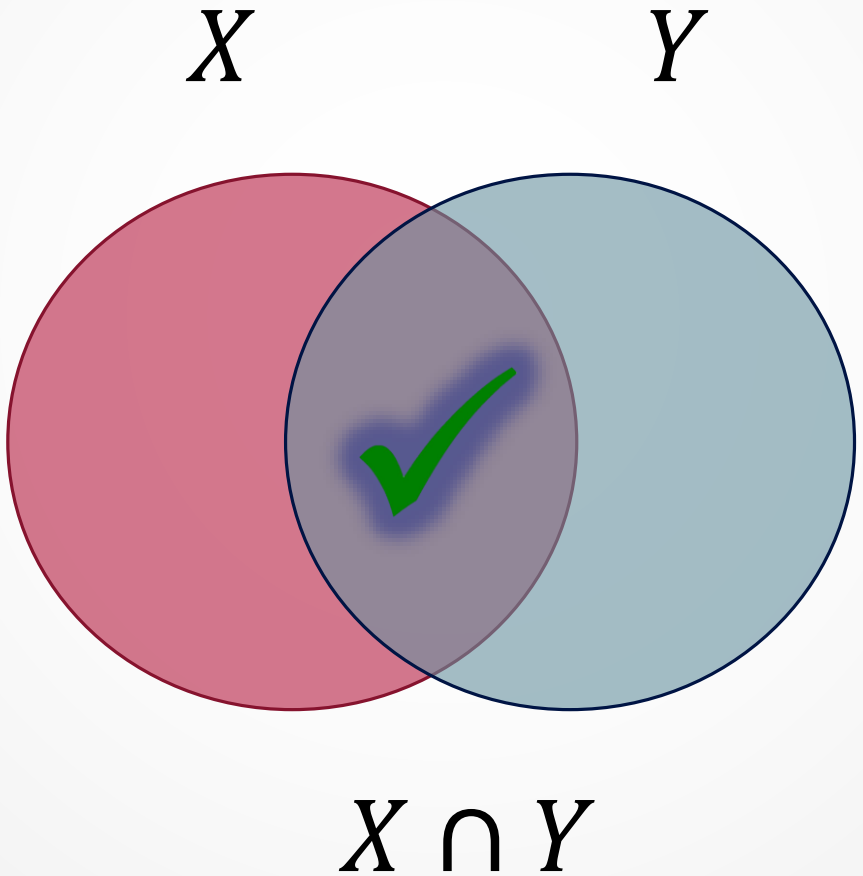
# Improved Private Set Intersection against Malicious Adversaries

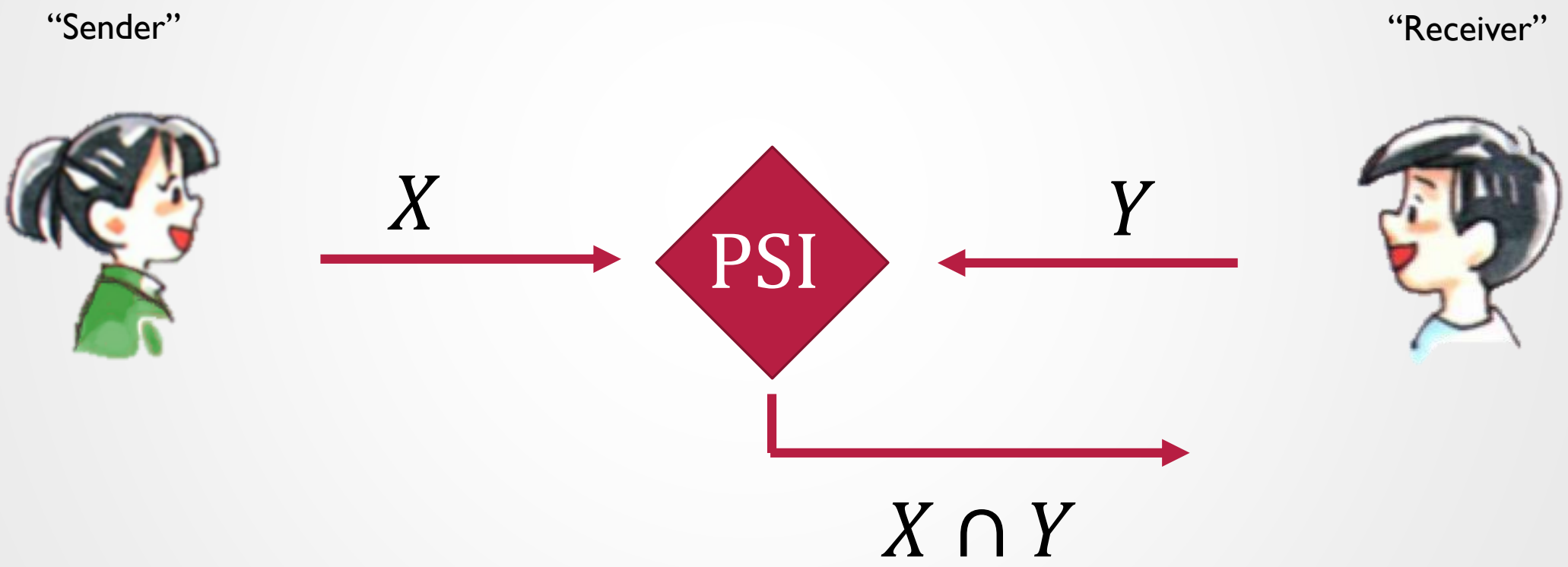
**Peter Rindal**
Mike Rosulek

Oregon State University OSU

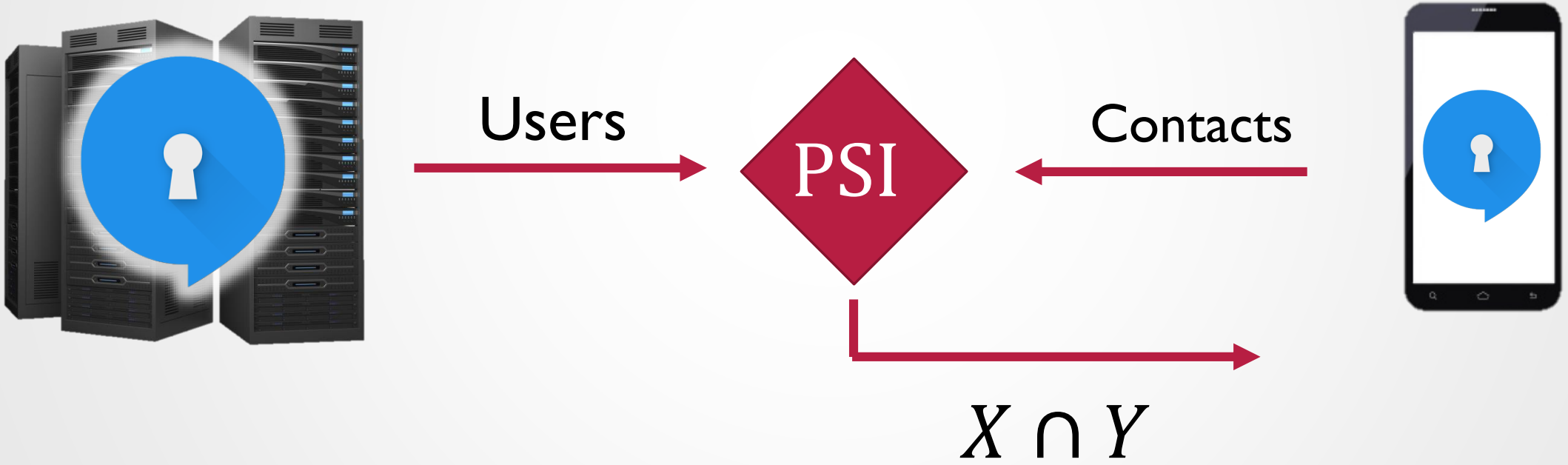
# Private Set Intersection (PSI)



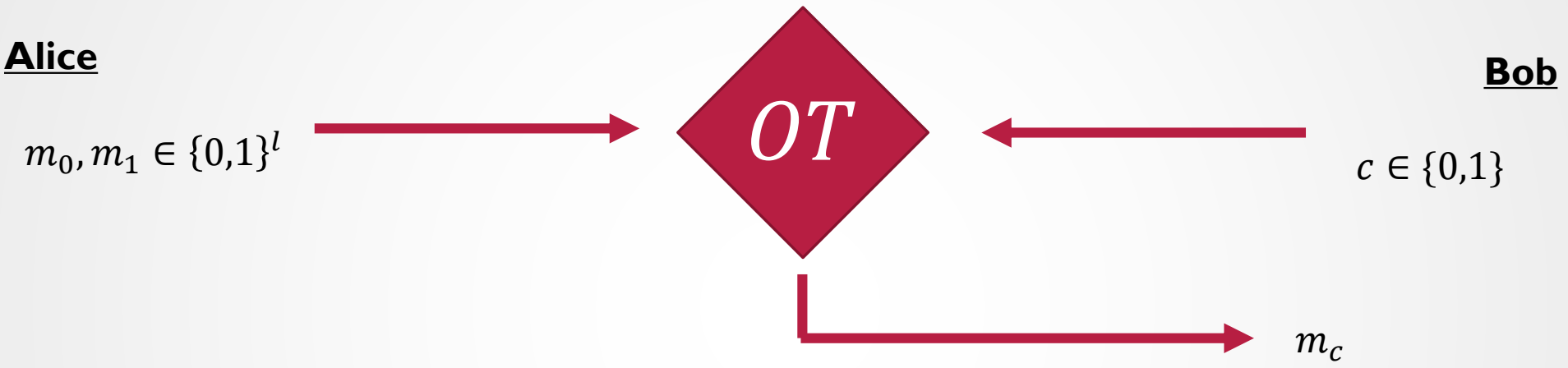

# Private Set Intersection (PSI)



# App: Contact discovery



# Oblivious Transfer (OT)




- Highly efficient and secure protocols exists
- Motivates it use as the basis for PSI

# Bloom Filter

Plain text data structure similar to hash table

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

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

$b =$

| | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

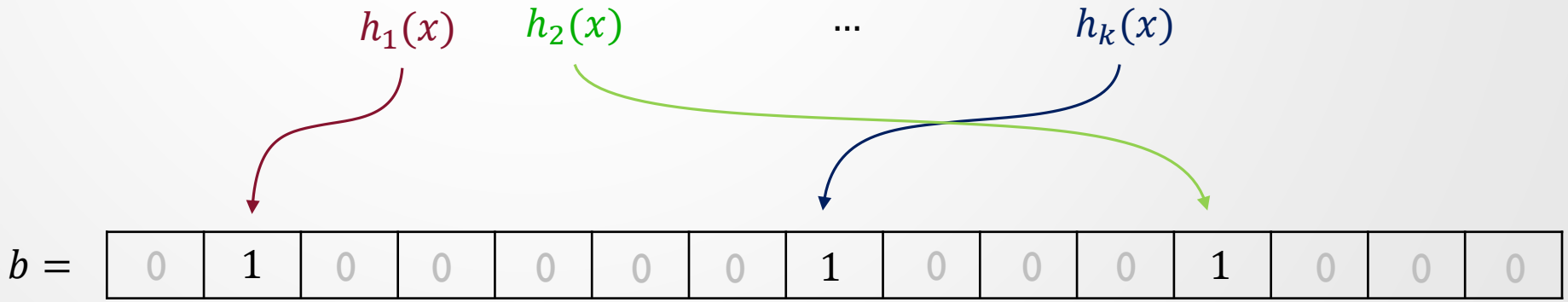
# Bloom Filter

Plai$n\ i\ (x) = 1 \quad , \forall i$

ext data structure similar to hash table

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

- To insert $x$, set
  - $b\ \ h\ i\ h\ i\ (x)\ = 1 \quad , \forall i$



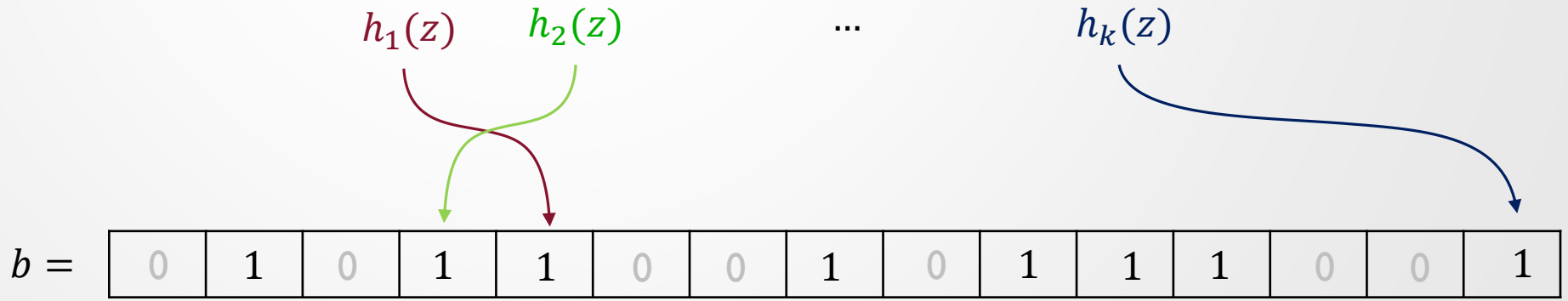
# Bloom Filter

Plai$n\ i\ (x) = 1 \quad , \forall i$

ext data structure similar to hash table

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

- To insert $x$, set
  - $b\ \ h\ i\ h\ i\ (x)\ = 1 \quad , \forall i$



# Bloom Filter

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

- To insert $x$, set
  - $b[h_i(x)] = 1 \quad , \forall i$
- To test membership
  - Return $\wedge_i \, b[h_i(x)]$

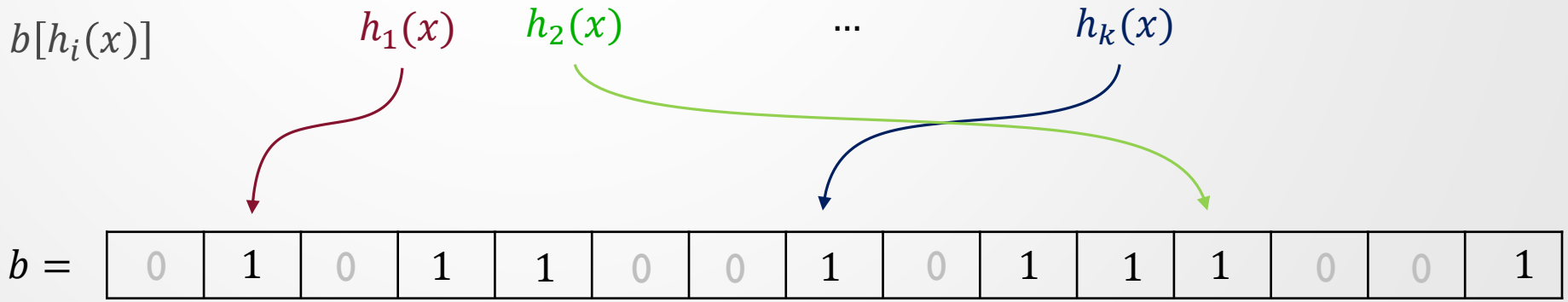
$b =$

| | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 1 |

# Bloom Filter

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

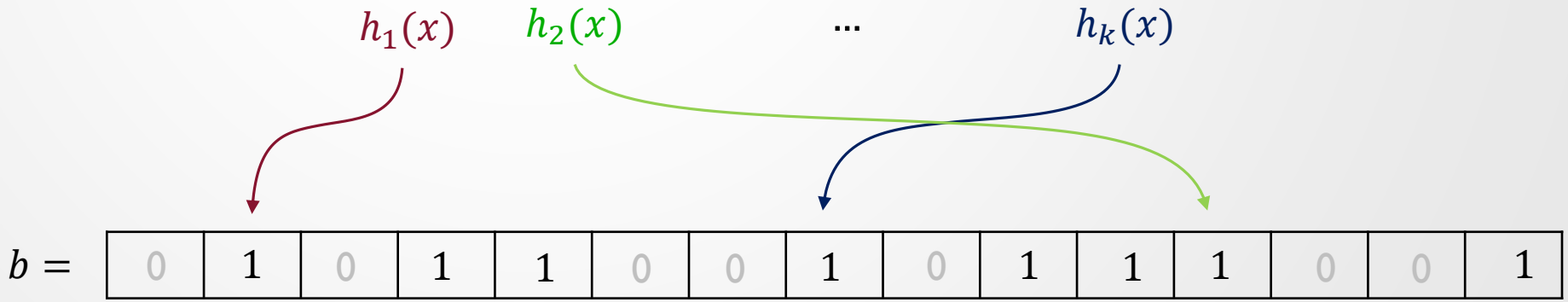
- To insert $x$, set
  - $b[h_i(x)] = 1 \quad , \forall i$
- To test membership
  - Return $\wedge_i \, b[h_i(x)]$



# Bloom Filter

$n$ items $\rightarrow$ Bloom filter with $m$ slots and $k$ hash functions

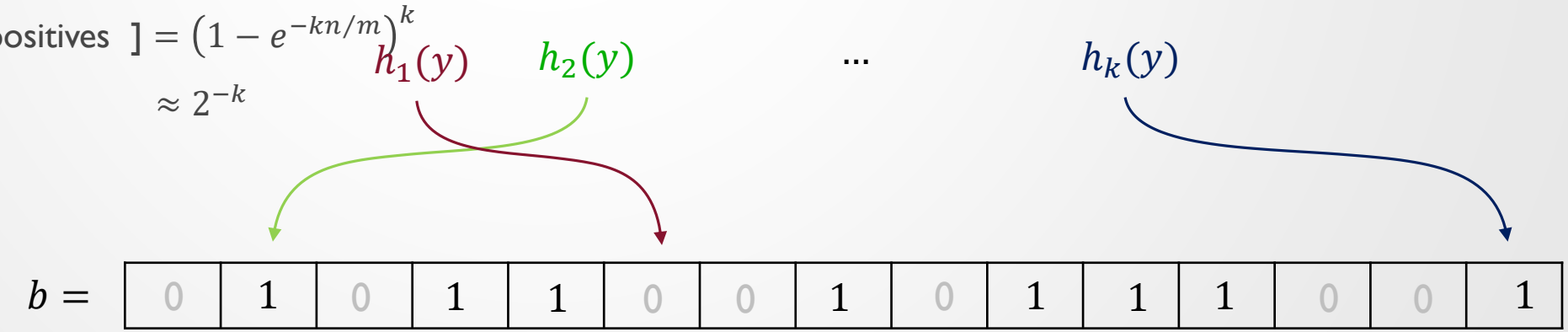
- Membership:
  - Pr[ false negatives ] = 0
  - $$\left(1 - e^{-kn/m}\right)^k$$
    $$\approx 2^{-k}$$



# Bloom Filter

$kn\,m\,kknn\,kn\,m\,mm\,kn\,m\;\;e\;-\;kn\,m\;\;1-\;e\;-\;kn\,m\;\;\;1-\;e\;-\;kn\,m\;\;\;k\,kk\;1-\;e\;-\;kn\,m\;\;\;k\;\;e\;-\;kn\,m\;\;ee\,e\;-\;kn\,m\;--\;1-\;e\;-\;kn\,m\;\;\;k\;1-\;e\;-\;kn\,m\;\;1n$ items → Bloom filter with $m$ slots and $k$ hash functions

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



# Bloom Filter

$\approx 2\ -k\ 2\ 2\ -k\ -kk\ 2\ -k$

$\approx 2\ -k\ 2\ 2\ -k\ -kk\ 2\ -k$

$kn\ m\ kknn\ kn\ m\ mm\ kn\ m\ \ e\ -\ kn\ m\ \ 1-\ e\ -\ kn\ m\ \ \ 1-\ e\ -\ kn\ m\ \ \ k\ kk\ 1-\ e\ -\ kn\ m\ \ \ k\ \ e\ -$
$kn\ m\ \ ee\ e\ -\ kn\ m\ --\ 1-\ e\ -\ kn\ m\ \ \ k\ 1-\ e\ -\ kn\ m\ \ 1n$ items → Bloom filter with $m$ slots and $k$ hash functions

- Membership:

$\approx 2^{-k}\ \ \approx 2^{-k}$

$h_1(y)$ $\approx 2^{-k}$ $h_2(y)$ … $h_k(y)$

$b =$

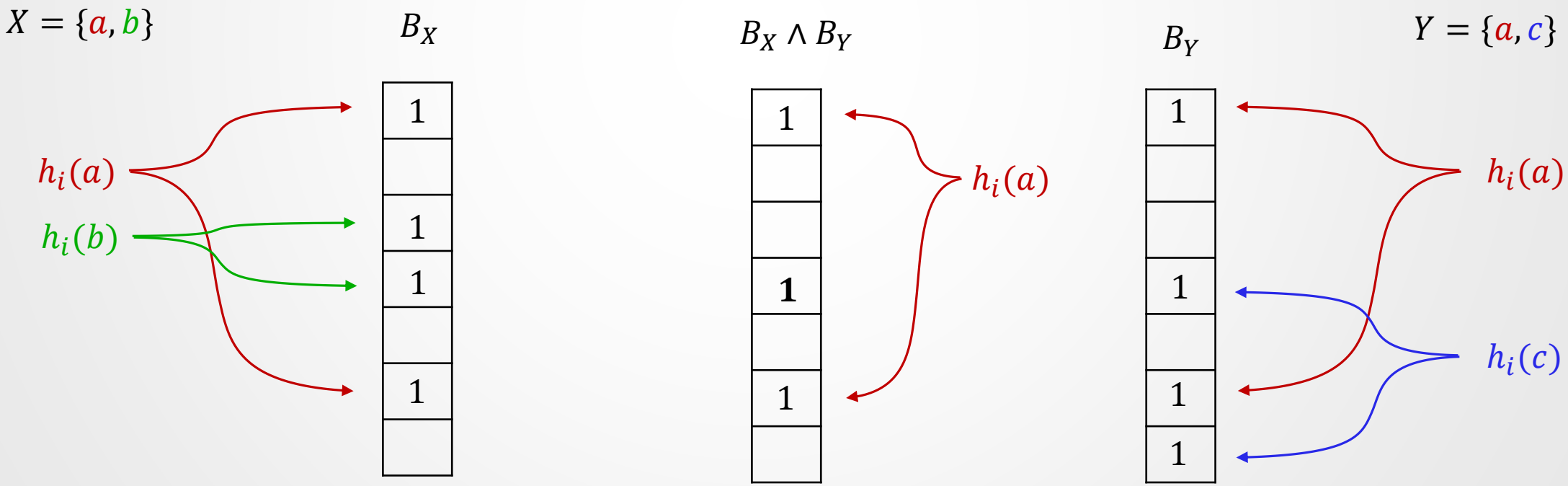
| | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 1 |

# Bloom Filter Intersection

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

# Bloom Filter Intersection

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



# Bloom Filter Protocol

[DongChenWen13, PinkasSchniederZohner14]

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

| |
|---|
| $m_0$ |
| $m_1$ |
| $m_2$ |
| $m_3$ |
| $m_4$ |
| $m_5$ |
| $m_6$ |

$B_Y$

$Y = \{a, c\}$

| |
|---|
| 1 |
| |
| |
| 1 |
| |
| 1 |
| 1 |

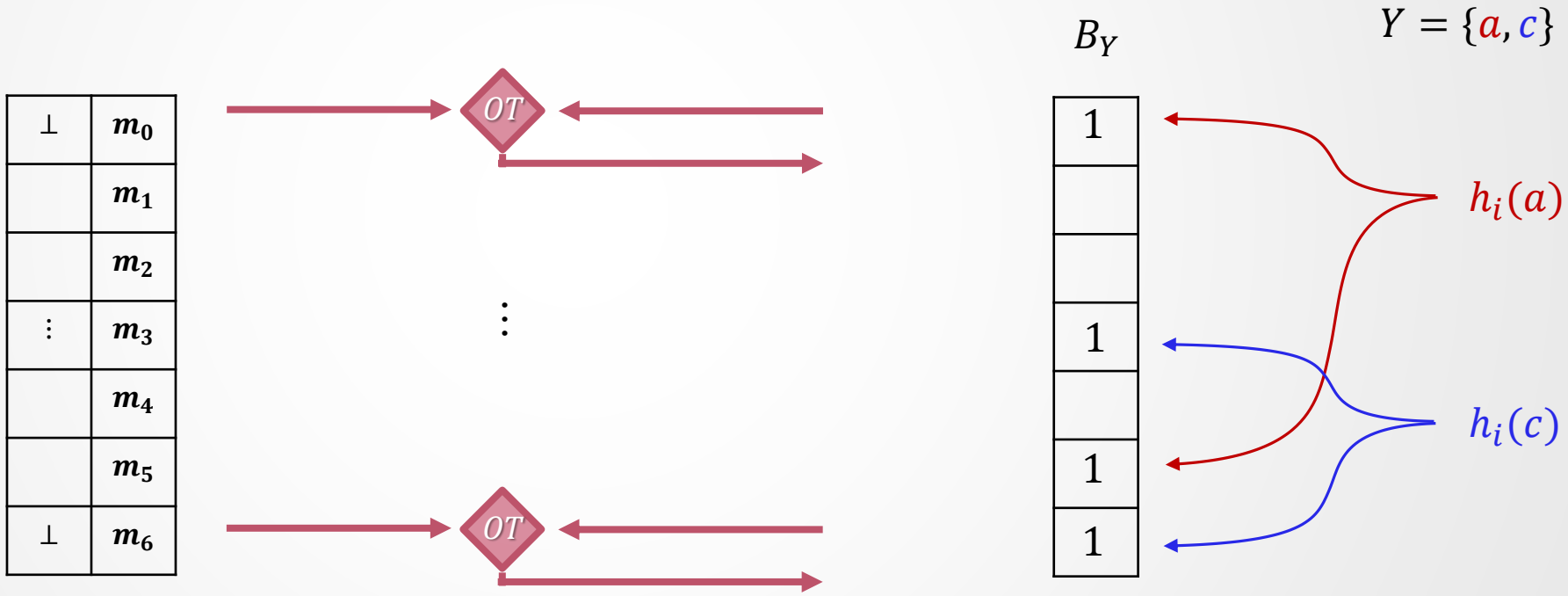
$h_i(a)$

$h_i(c)$

# Bloom Filter Protocol

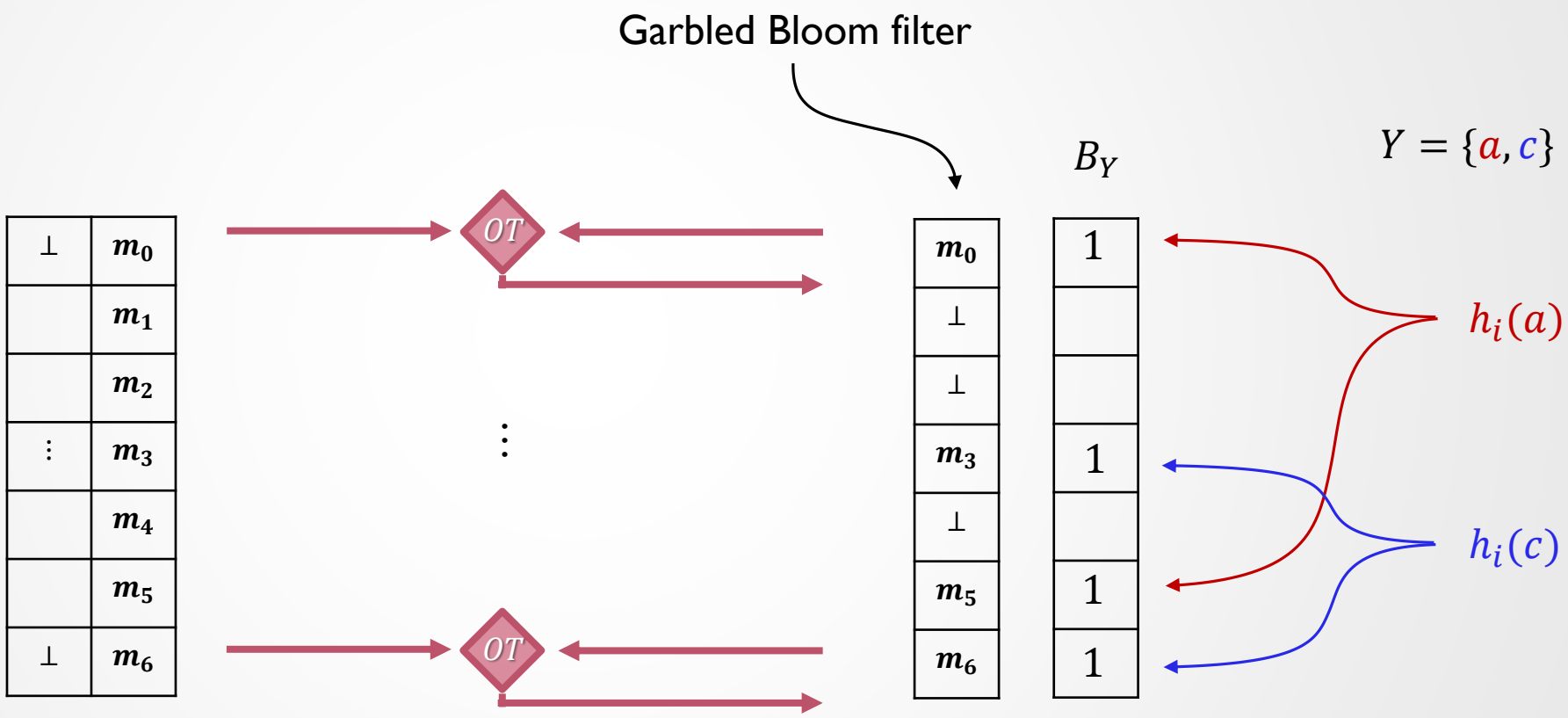
[DongChenWen13, PinkasSchniederZohner14]

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



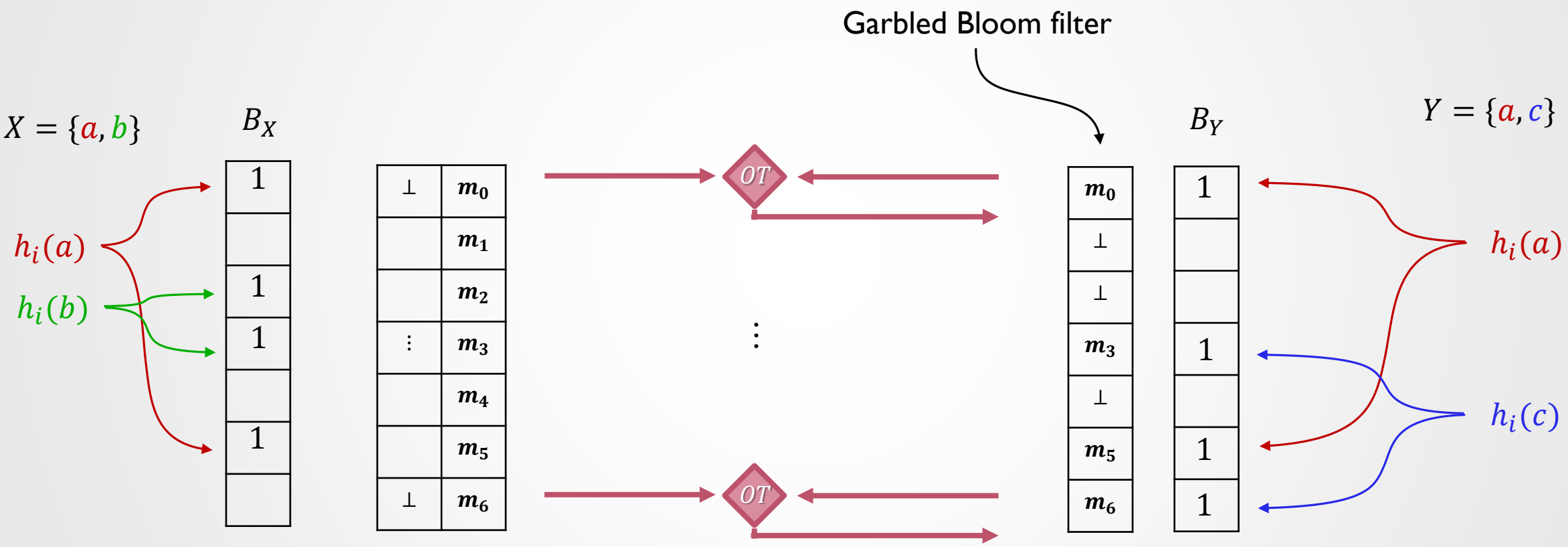
# Bloom Filter Protocol

[DongChenWen13, PinkasSchniederZohner14]



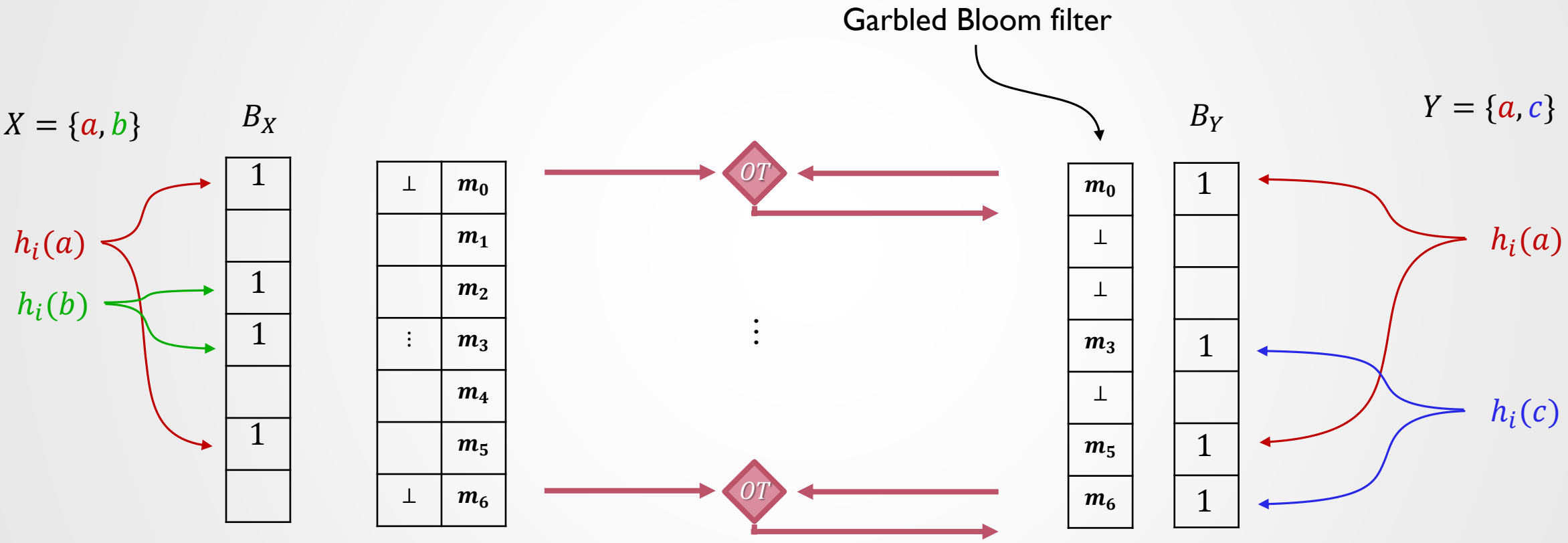
# Bloom Filter Protocol

[DongChenWen13, PinkasSchniederZohner14]



# Bloom Filter Protocol

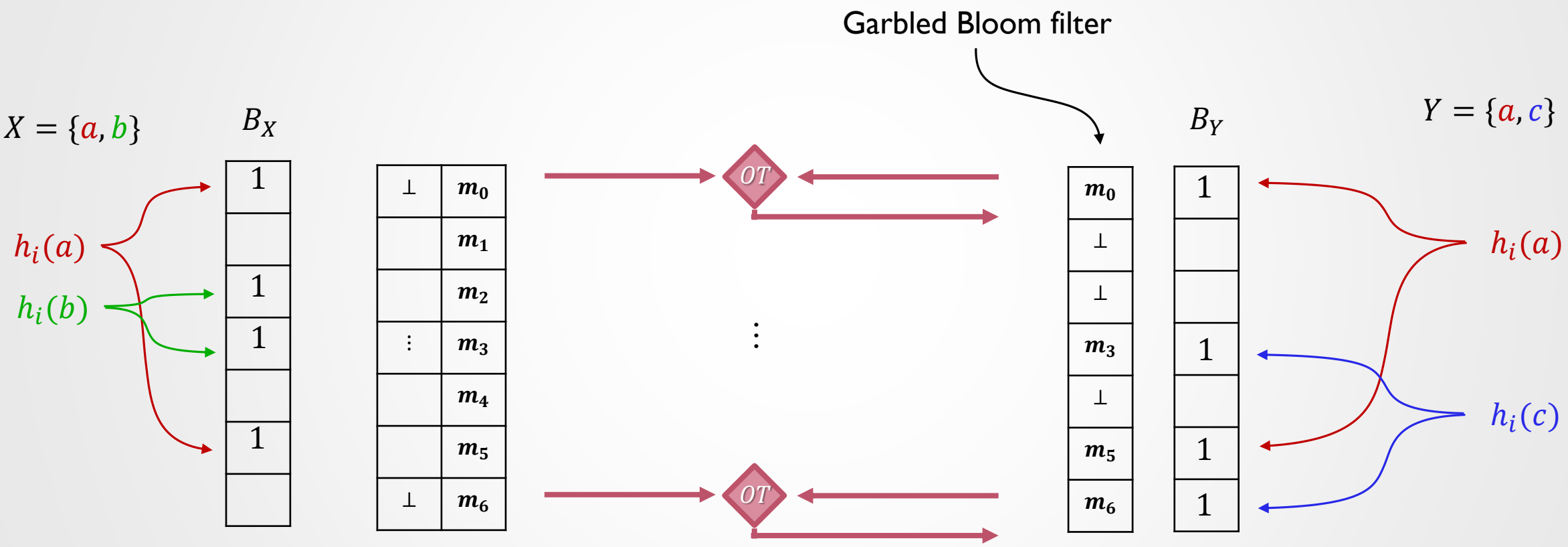
[DongChenWen13, PinkasSchniederZohner14]




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

# Bloom Filter Protocol

[DongChenWen13, PinkasSchniederZohner14]




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

Output the intersection

$$\widehat{X} \cap \{m_0 \oplus m_5, m_3 \oplus m_6\}$$

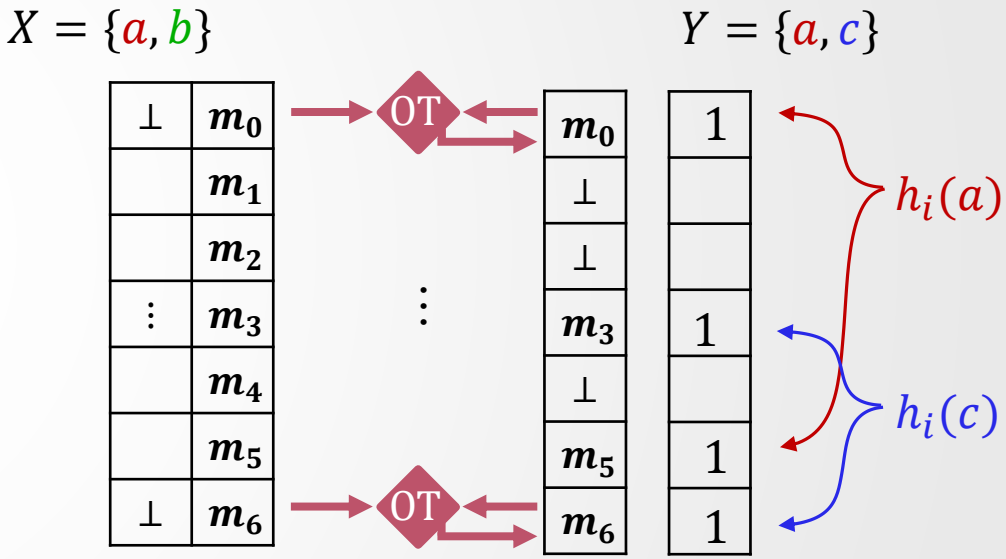
# Semi-Honest Security

[DongChenWen13, PinkasSchniederZohner14]

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[ false positives ] $\approx 2^{-k}$




Output:

$$\hat{X} = \begin{Bmatrix} m_0 \oplus m_5, \\ m_2 \oplus m_3 \end{Bmatrix} \longrightarrow \hat{X} \cap \begin{Bmatrix} m_0 \oplus m_5, \\ m_3 \oplus m_6 \end{Bmatrix}$$

# Semi-Honest Security

[DongChenWen13, PinkasSchniederZohner14]

$y' \notin YY$, Receiver learns encoding

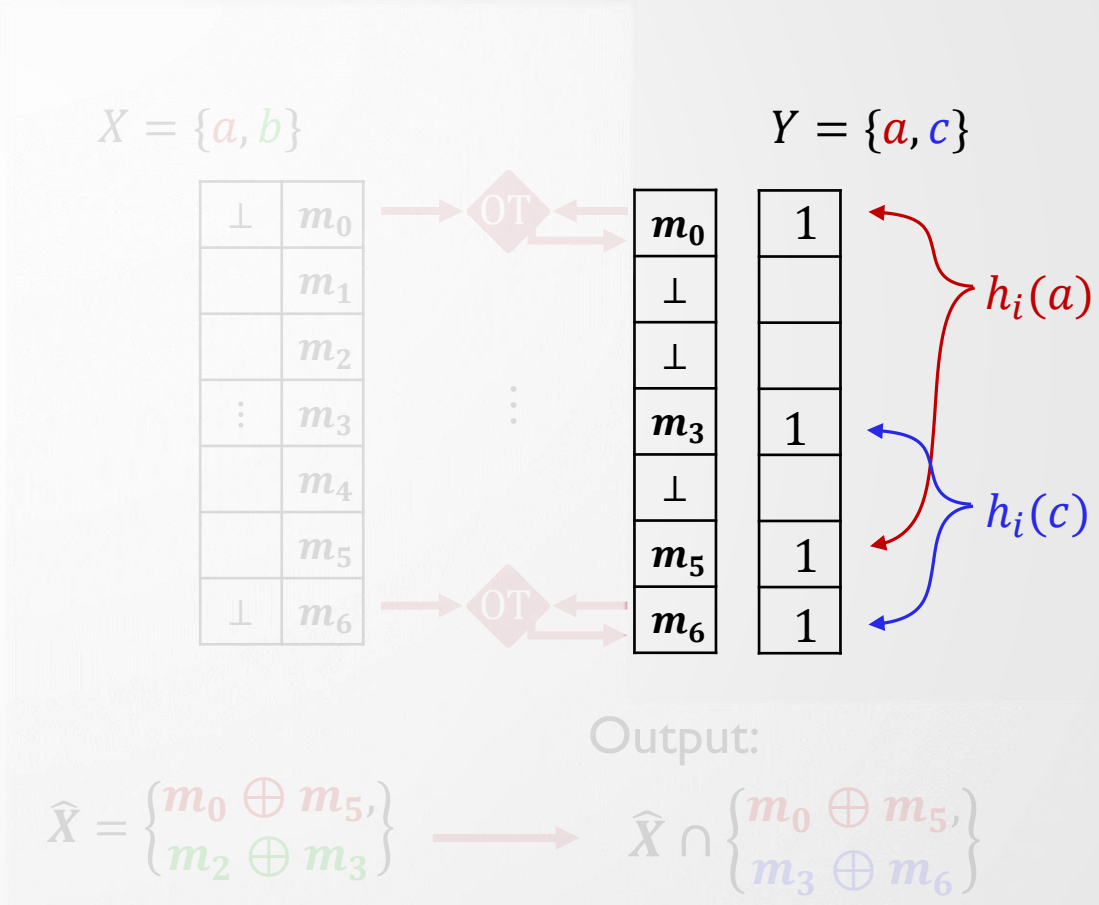
Naturally secure against Sender.

- 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[ false positives ] $\approx 2^{-k}$



# Semi-Honest Security

[DongChenWen13, PinkasSchniederZohner14]

e.g. Encode $y'$ $yy'$ $y'$ = $m$ 3 $mm$ $m$ 3 3 $m$ 3 $\oplus$ $m$ 4 $mm$ $m$ 4 4 $m$ 4

$y' \notin YY$, Receiver learns encoding

Naturally secure against Sender.

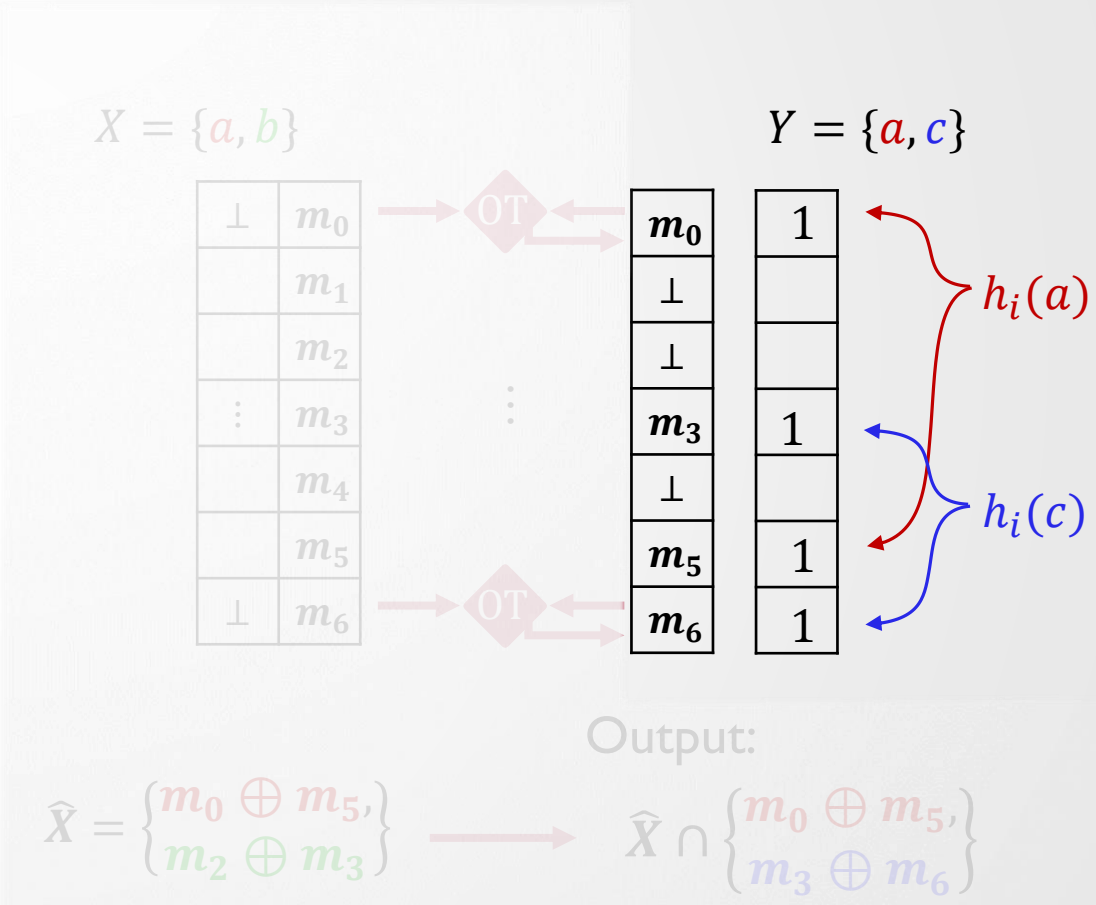
- OT hides select bits
- Final message sent to Receiver

- Secure against Receiver

$\text{ncode}(y') = m_3 \oplus m_4$ e.g.

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

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



# Semi-Honest Security

[DongChenWen13, PinkasSchniederZohner14]

e.g. Encode $y'$ $yy'$ $y'$ = $m$ 3 $mm$ $m$ 3 3 $m$ 3 $\oplus$ $m$ 4 $mm$ $m$ 4 4 $m$ 4

$y' \notin YY$, Receiver learns encoding

Naturally secure against Sender.

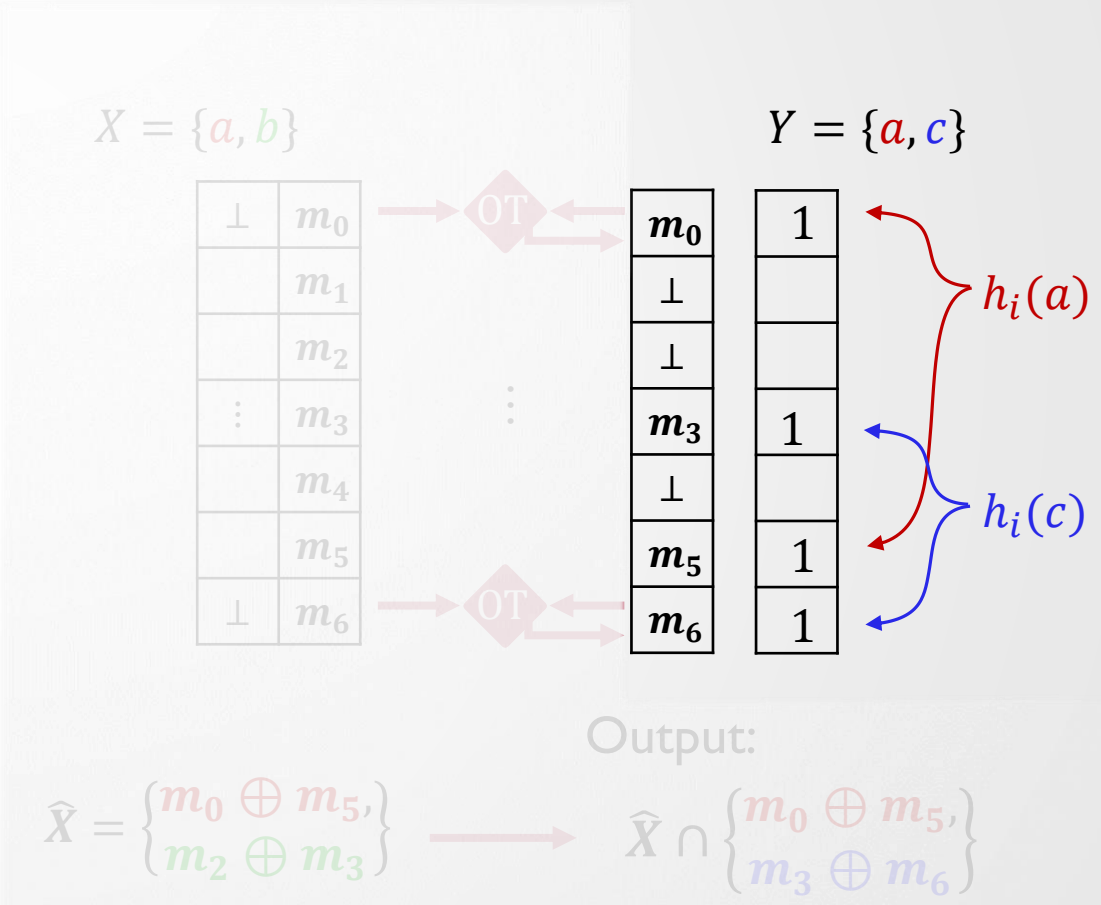
- OT hides select bits
- Final message sent to Receiver

- Secure against Receiver

$\text{ncode}(y') = m_3 \oplus m_4$ e.g.

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

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



# Semi-Honest Security

[DongChenWen13, PinkasSchniederZohner14]

$2\ -k\ 2\ 2\ -k\ -kk\ 2\ -k$

e.g. Encode $y'\ yy'\ y' = m\ 3\ mm\ m\ 3\ 3\ m\ 3 \oplus m\ 4\ mm\ m\ 4\ 4\ m\ 4$

$y\ ' \notin YY$, Receiver learns encoding

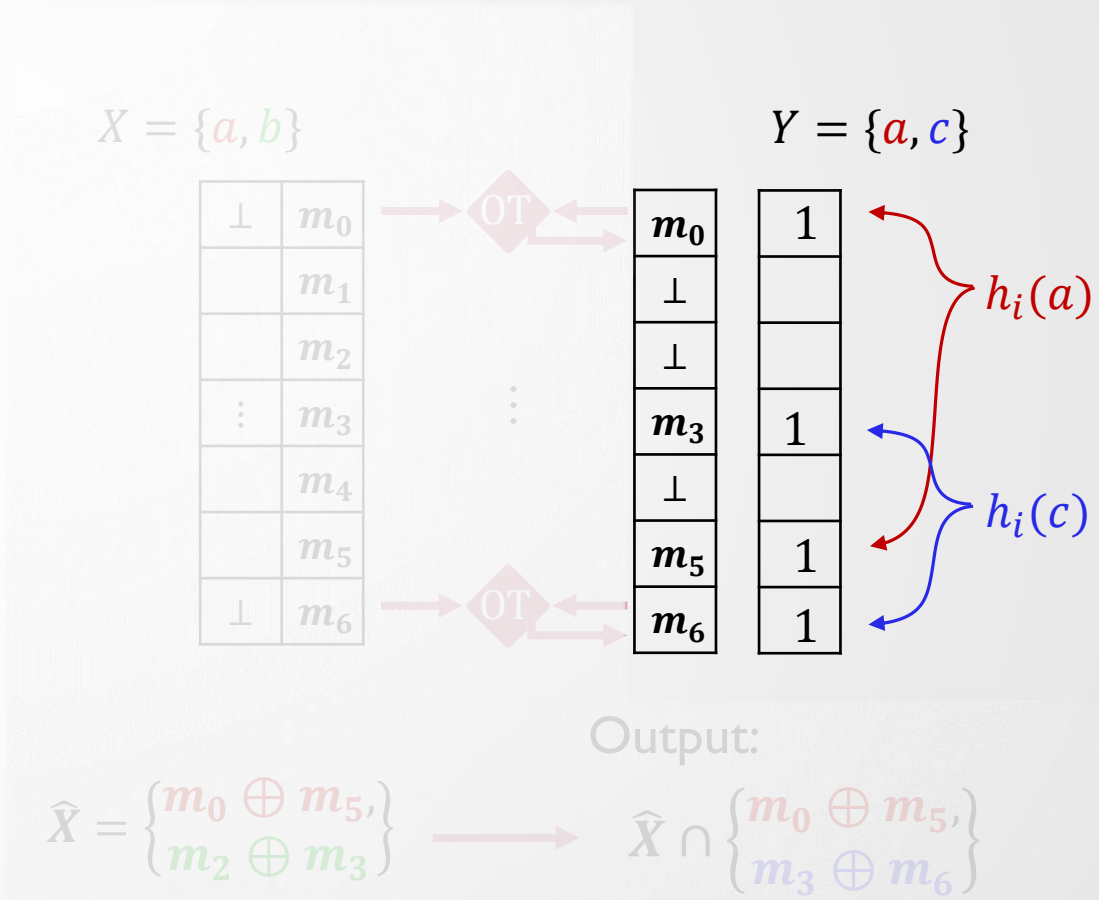
Naturally secure against Sender.

- OT hides select bits
- Final message sent to Receiver

- Secure against Receiver

  $\mathrm{ncode}(y') = m_3 \oplus m_4$ e.g. $\mathrm{Encode}(y') = m_3 \oplus m_4$

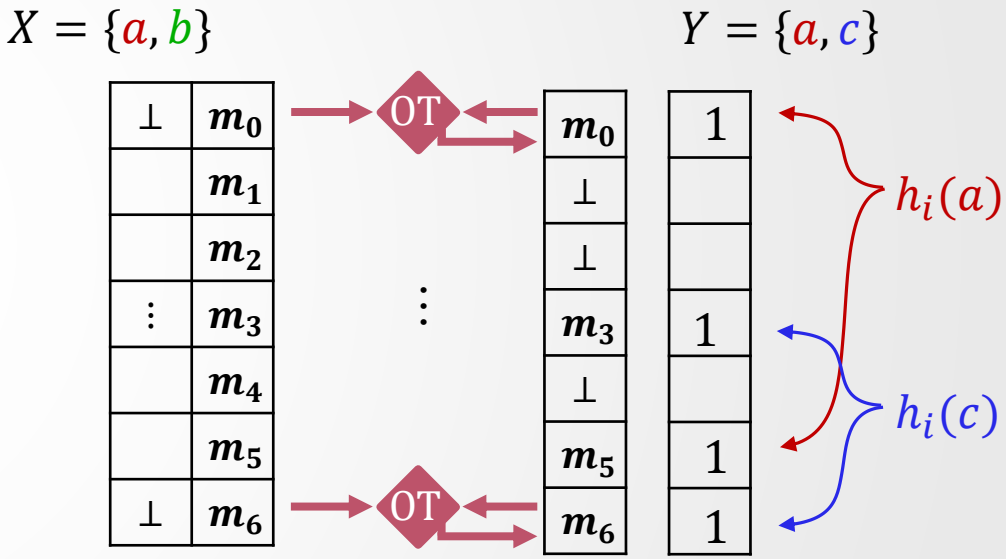
  - DCW13 show equivalence to false positive in standard bloom filter
    - Pr[ false positives ] $\approx 2^{-k}$
    - Pr[ false positives ] $\approx 2^{-k}$



# Malicious Receiver

**Insecure** against Receiver

- Bloom filter
- Receiver will obtain all $m_i$
- Can probe for $\boldsymbol{m_2 \oplus m_3}$




Output:

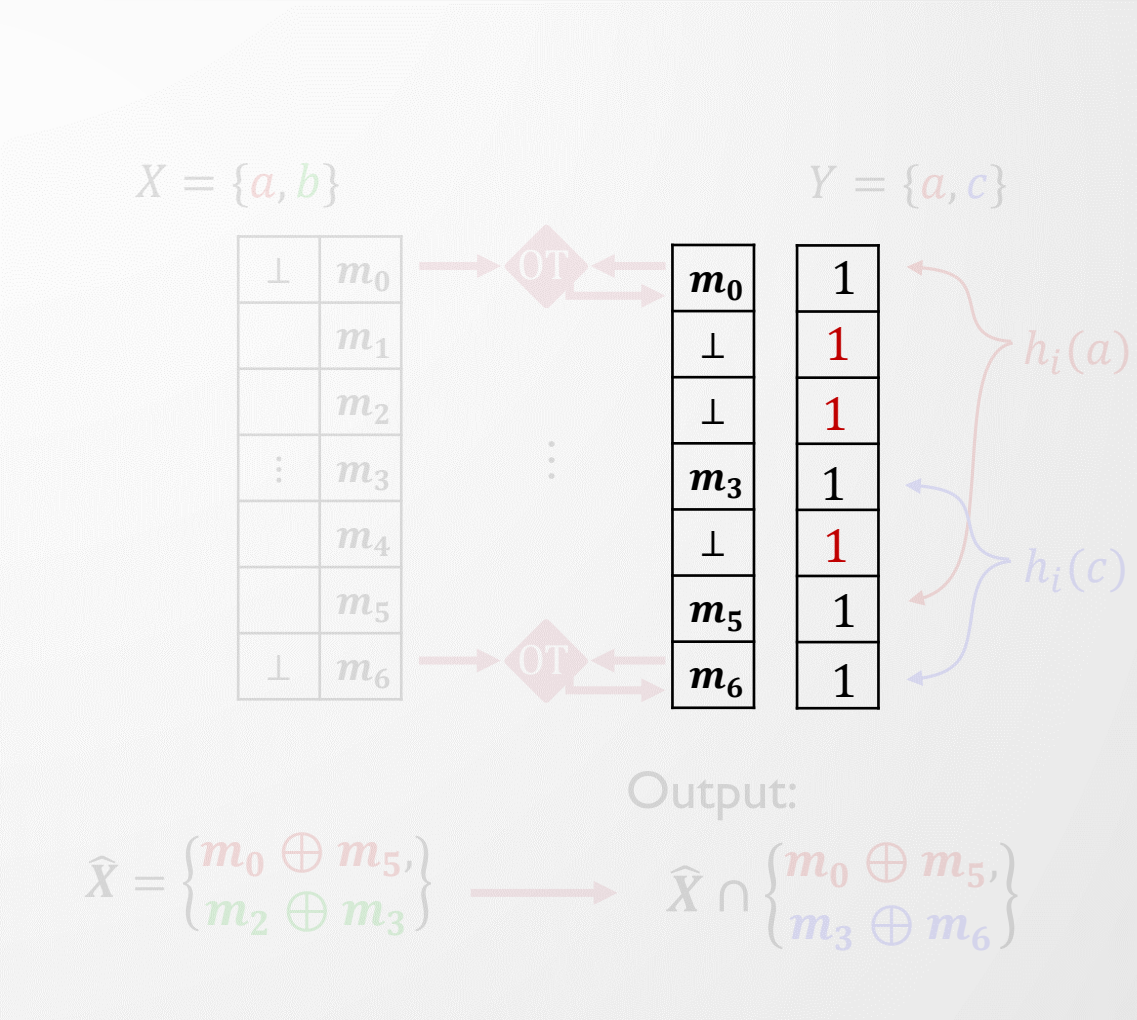
$$\hat{X} = \begin{Bmatrix} \boldsymbol{m_0 \oplus m_5}, \\ \boldsymbol{m_2 \oplus m_3} \end{Bmatrix} \longrightarrow \hat{X} \cap \begin{Bmatrix} \boldsymbol{m_0 \oplus m_5}, \\ \boldsymbol{m_3 \oplus m_6} \end{Bmatrix}$$

# Malicious Receiver

Bloom filter

Insecure against Receiver

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

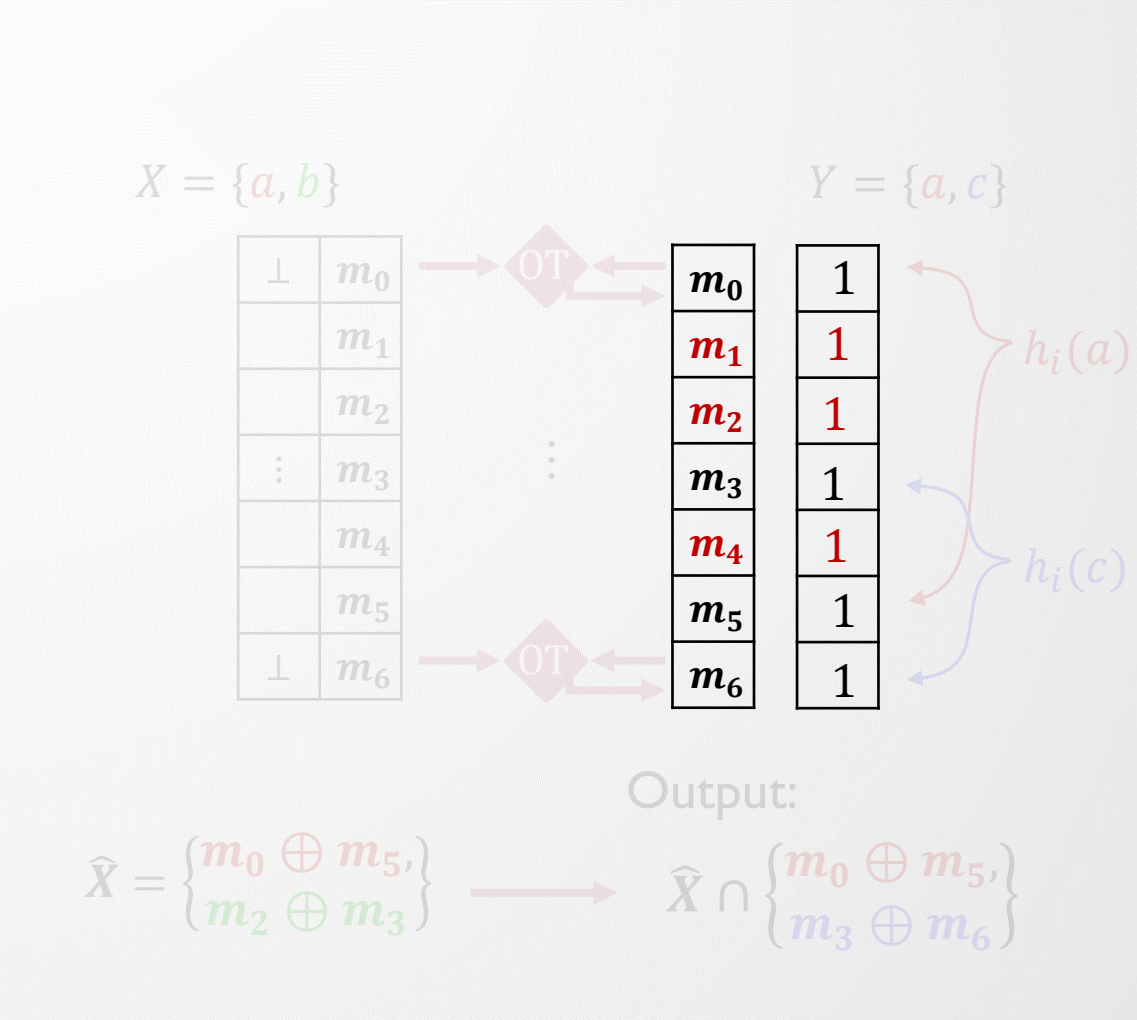


# Malicious Receiver

Bloom filter

Insecure against Receiver

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



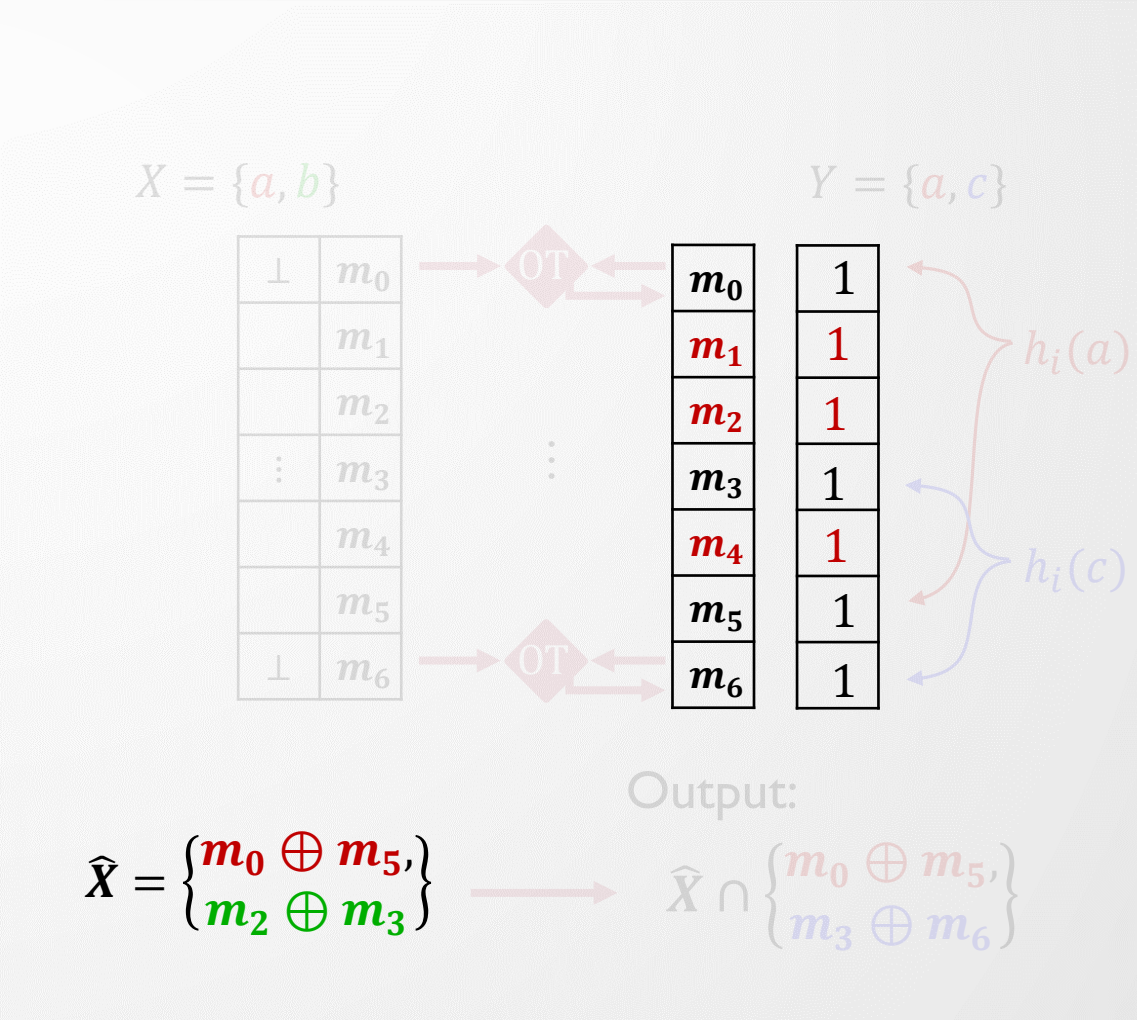
# Malicious Receiver

$\boldsymbol{m\ 3\ mm\ m\ 3\ 33\ m\ 3}$

Bloom filter

Insecure against Receiver

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

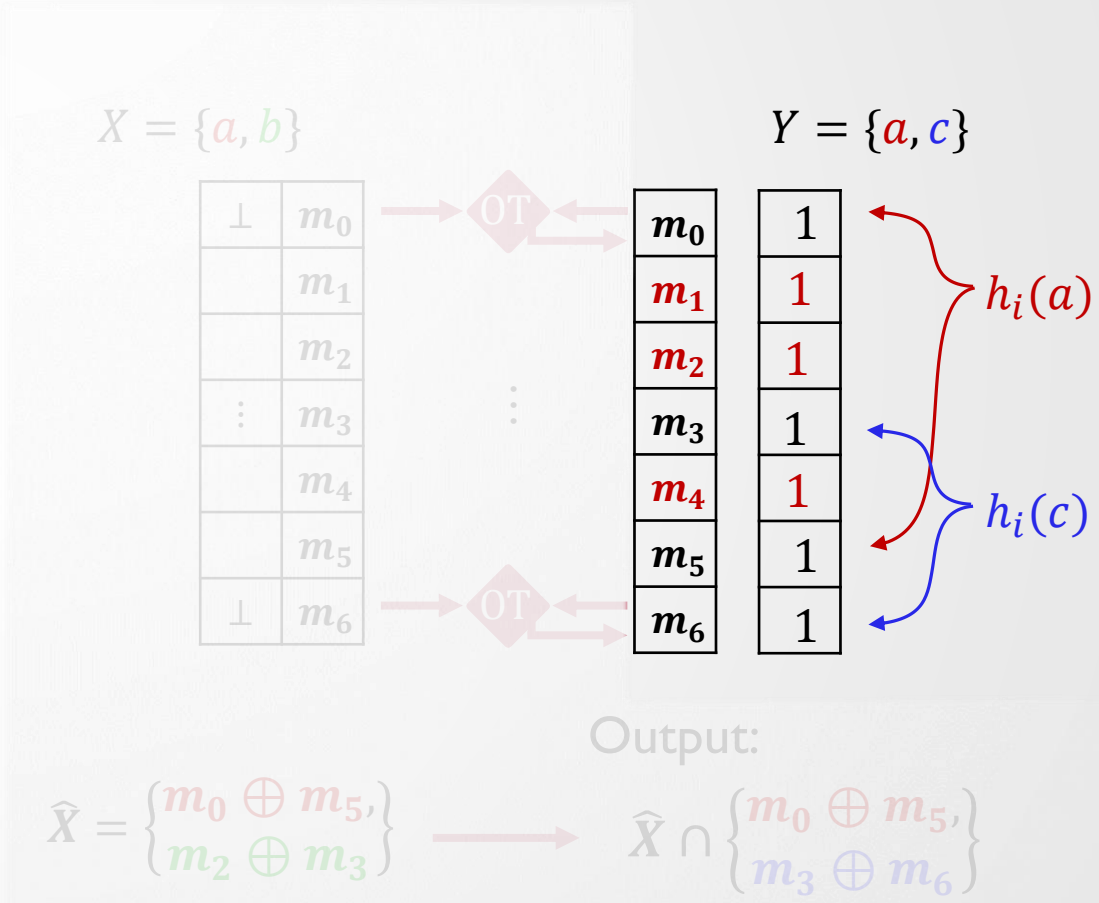


# Warm-Up: The DongChenWen13 Approach

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 $s$



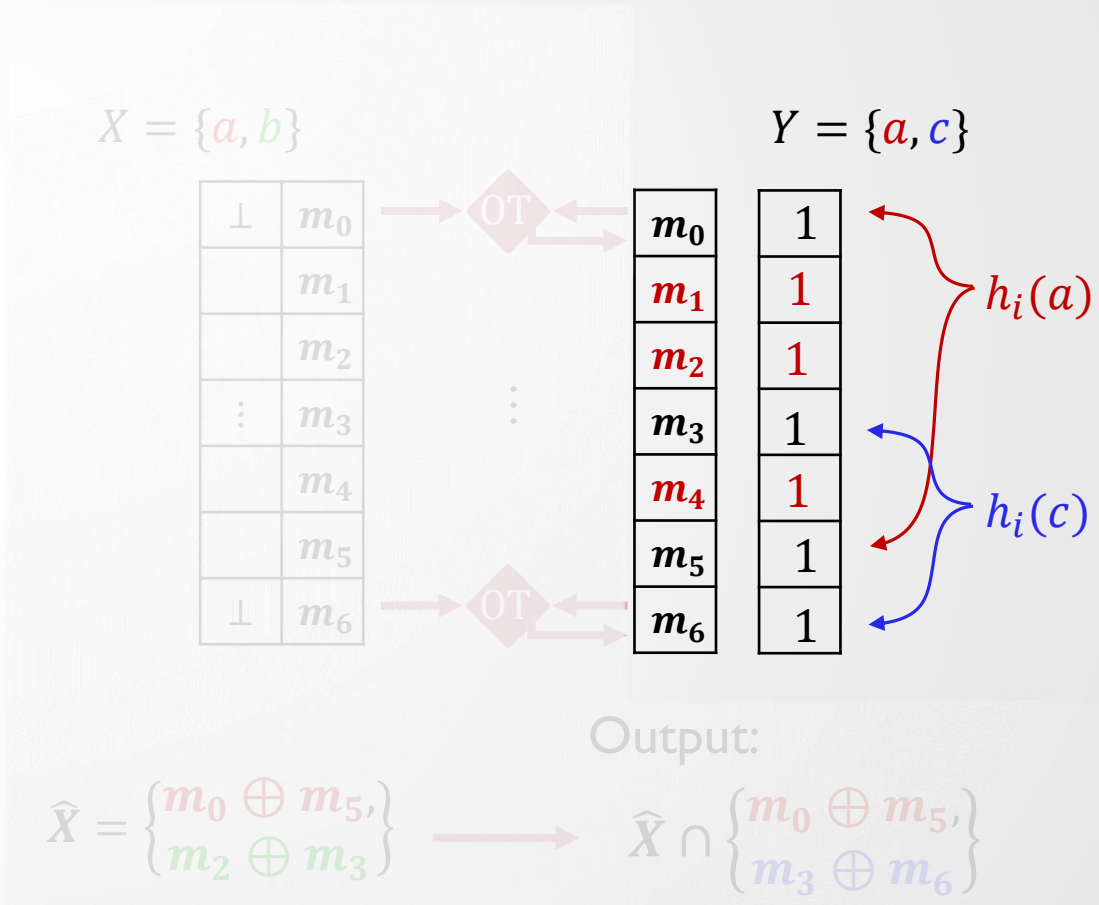
# Warm-Up: The DongChenWen13 Approach

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 $s$



# Warm-Up: The DongChenWen13 Approach

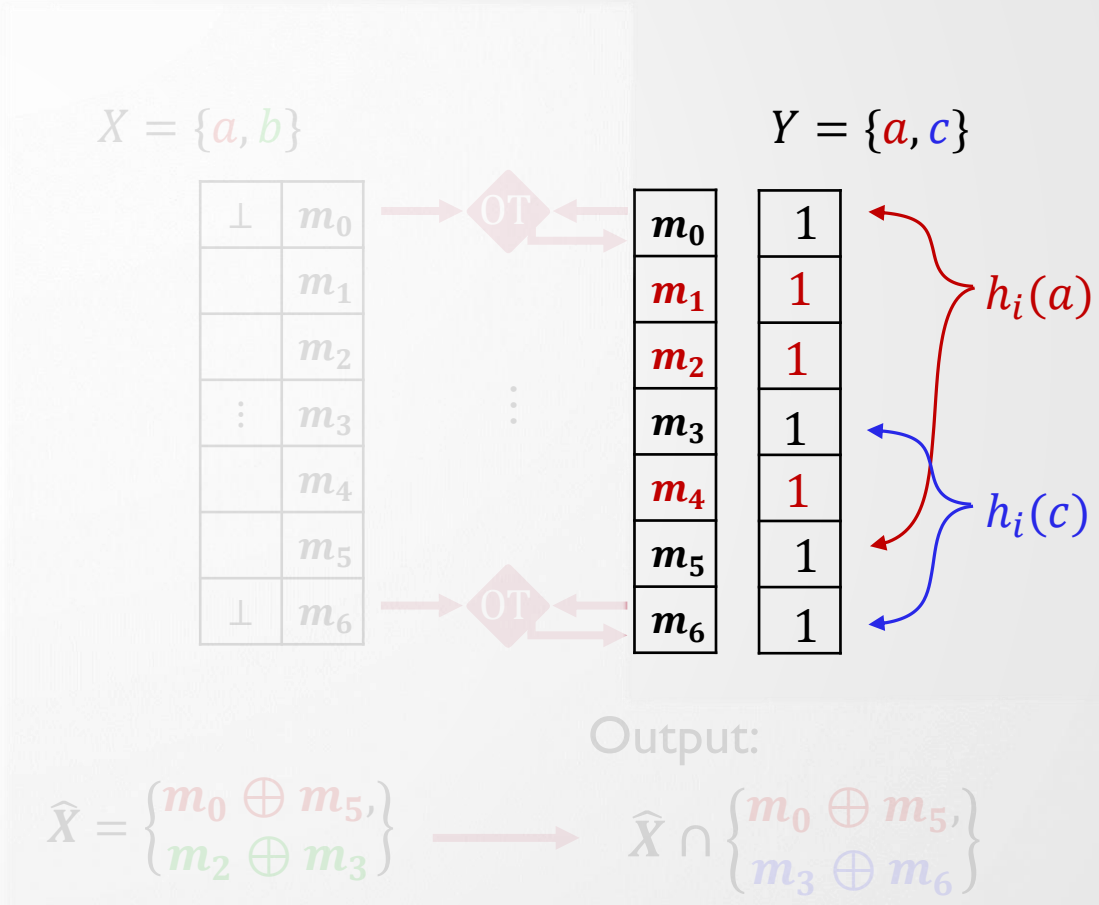
0,1 $\kappa$ 0,1 0,1 0,1 0,1 $\kappa$ $\kappa\kappa$ 0,1 $\kappa$

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
  - 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 $s$



# Warm-Up: The DongChenWen13 Approach

$s$ 1 ,…, $s$ $m$ $ss$ $s$ $m$ $mm$ $s$ $m$

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

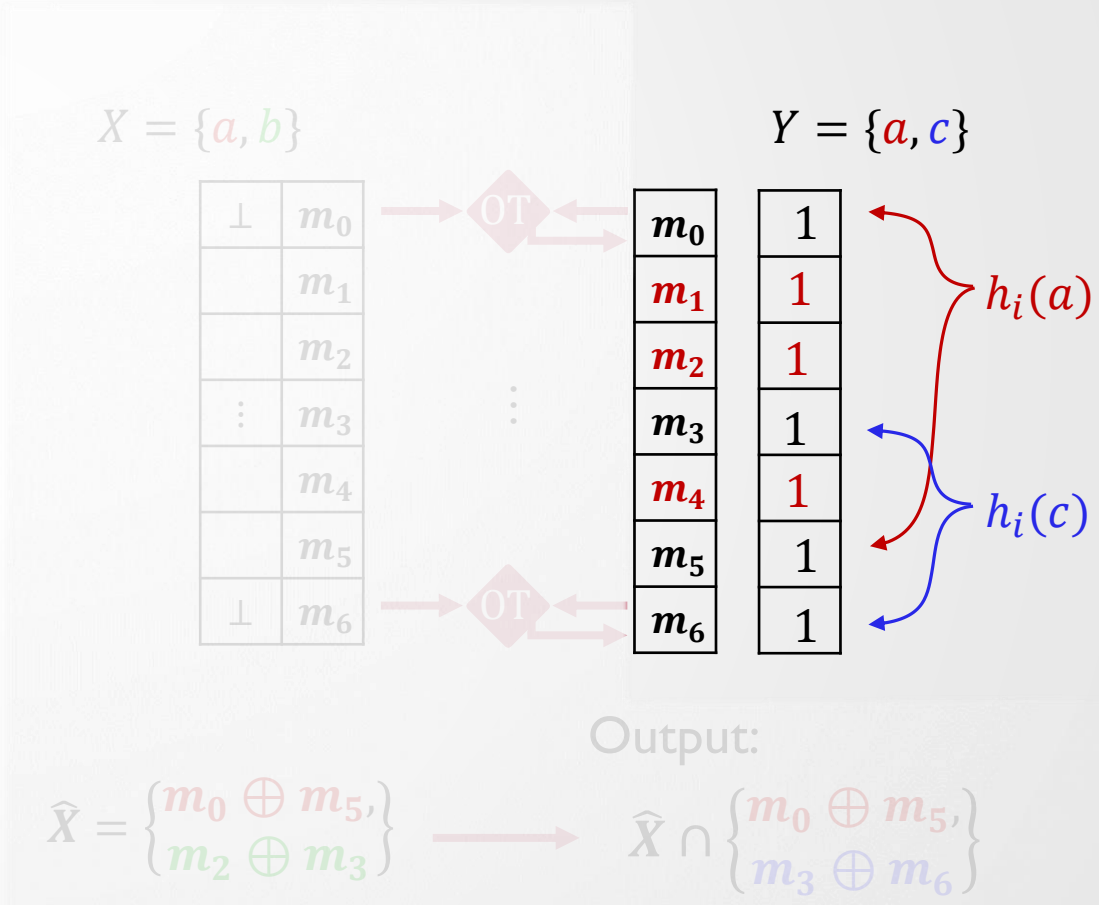
0,1 $\kappa$ 0,1 0,1 0,1 0,1 $\kappa$ $\kappa\kappa$ 0,1 $\kappa$

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 1 , … , $s_m$
  - 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 $ii$th zero OT message

$s$ 1 ,…, $s$ $m$ $ss$ $s$ $m$ $mm$ $s$ $m$

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

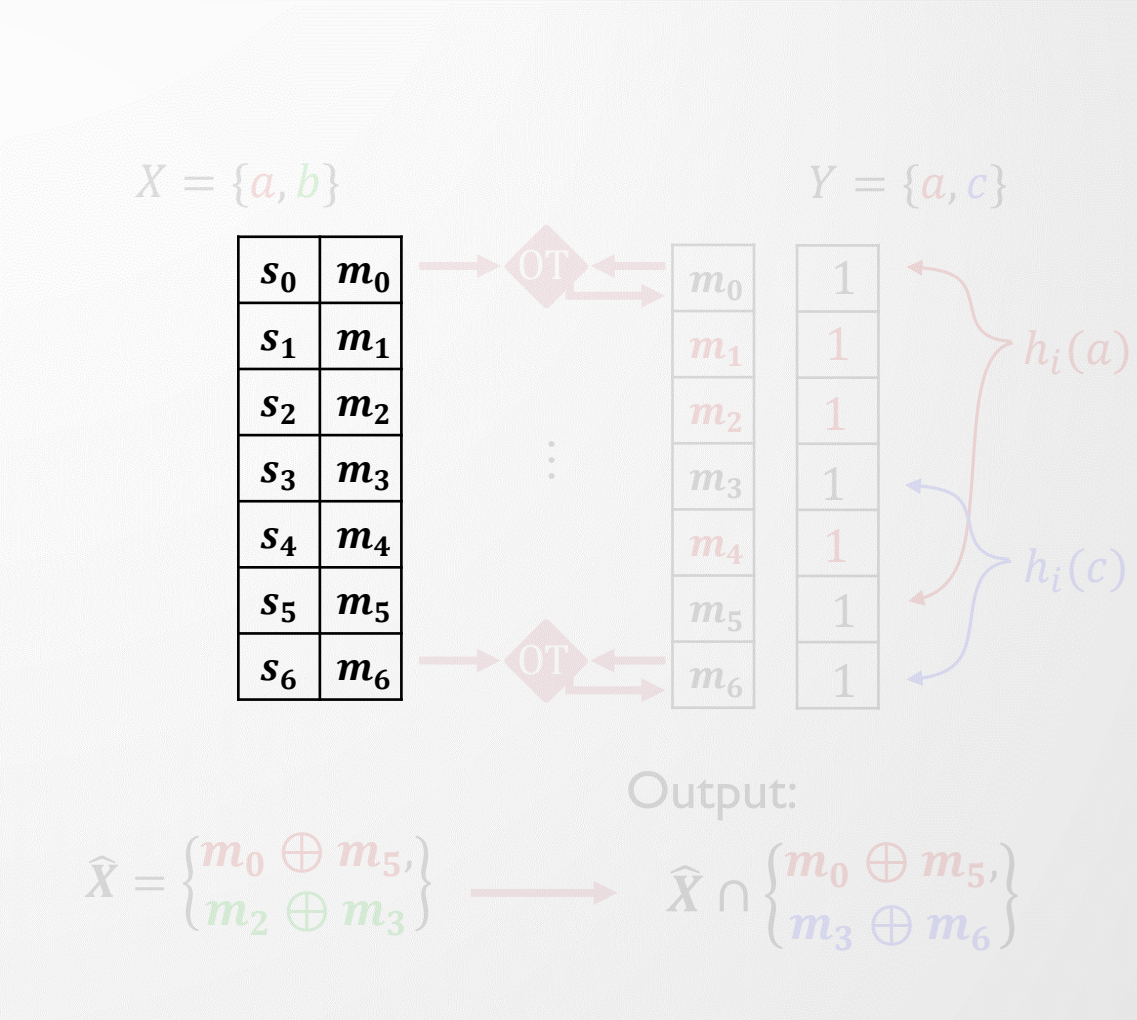
0,1 $\kappa$ 0,1 0,1 0,1 0,1 $\kappa$ $\kappa\kappa$ 0,1 $\kappa$

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 1 , … , $s_m$
  - Transmit $s$ $i$ $i$ $i$ $i$ as the $i$th zero OT message
  - Generate a $\frac{m}{2}$ out of $m$ secret sharing of $s$
    - $s_1, \dots, s_m$
  - Transmit $s_i$ as the $i$th zero OT message
  - Encrypt summary values with $s$



# Warm-Up: The DongChenWen13 Approach

as the $ii$th zero OT message

$s$ 1 ,…, $s$ $m$ $ss$ $s$ $m$ $mm$ $s$ $m$

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

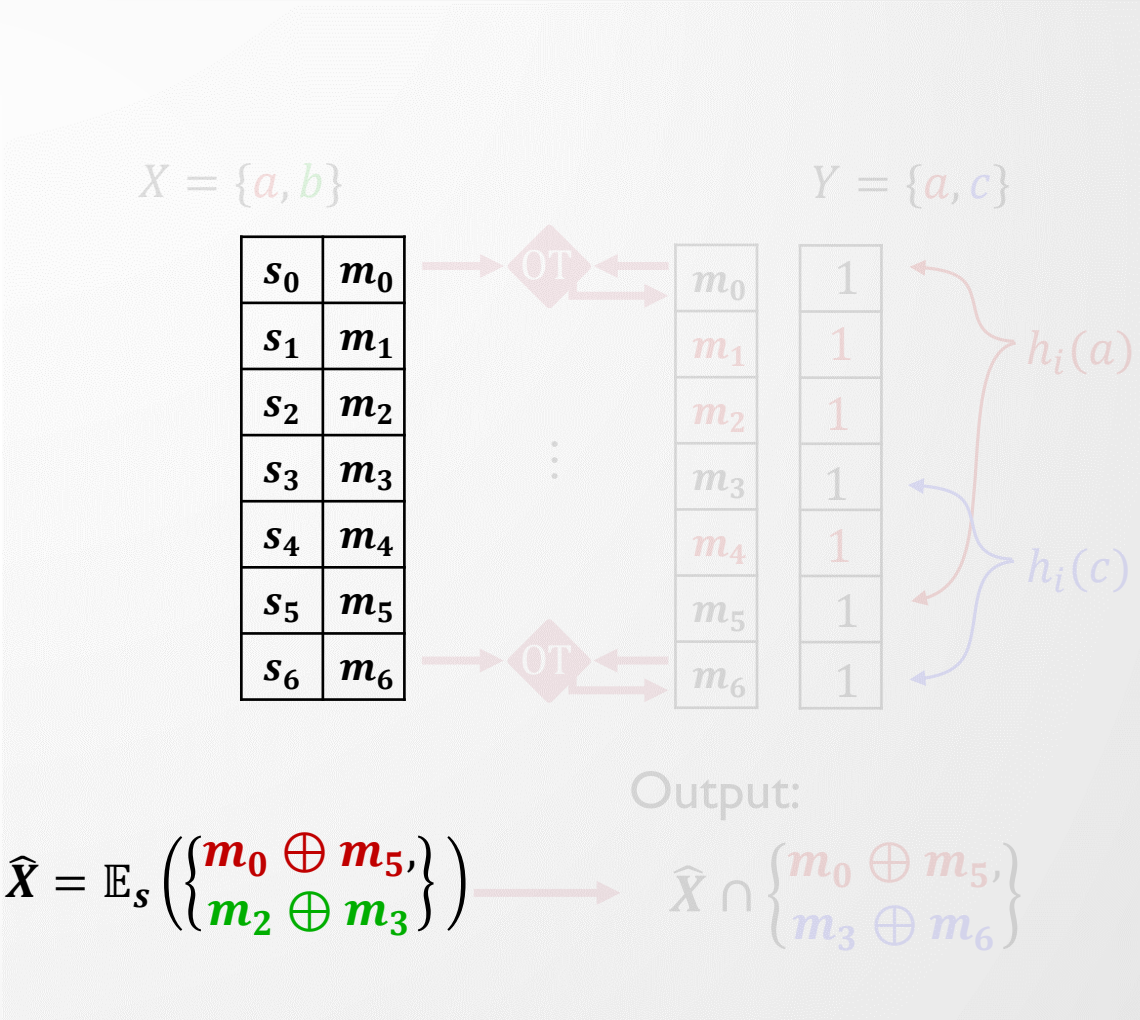
0,1 $\kappa$ 0,1 0,1 0,1 0,1 $\kappa$ $\kappa\kappa$ 0,1 $\kappa$

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 1 , … , $s_m$
  - Encrypt summary values with $s$Generate a $\frac{m}{2}$ out of $m$ secret sharing of $s$
    - $s_1, \dots, s_m$
  - Transmit $s_i$ as the $i$th zero OT message
  - Encrypt summary values with $s$



# Warm-Up: The DongChenWen13 Approach

as the $ii$th zero OT message

$s$ 1 ,…, $s$ $m$ $ss$ $s$ $m$ $mm$ $s$ $m$

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

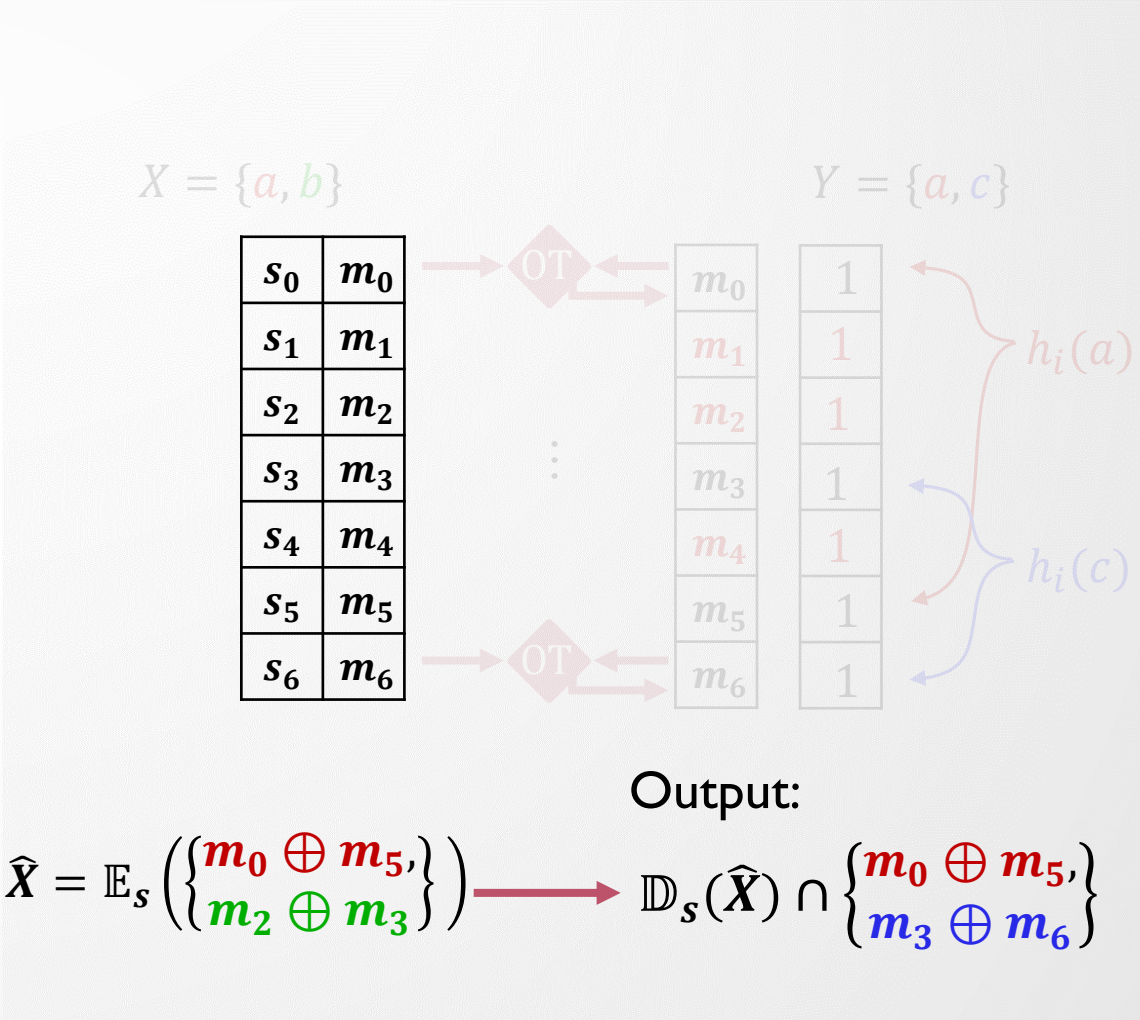
0,1 $\kappa$ 0,1 0,1 0,1 0,1 $\kappa$ $\kappa\kappa$ 0,1 $\kappa$

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 1 , … , $s_m$
  - Encrypt summary values with $s$Generate a $\frac{m}{2}$ out of $m$ secret sharing of $s$
    - $s_1, \dots, s_m$
  - Transmit $s_i$ as the $i$th zero OT message
  - Encrypt summary values with $s$



# Warm-Up: The DongChenWen13 Approach

as the $ii$th zero OT message

$s$ 1 ,…, $s$ $m$ $ss$ $s$ $m$ $mm$ $s$ $m$

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

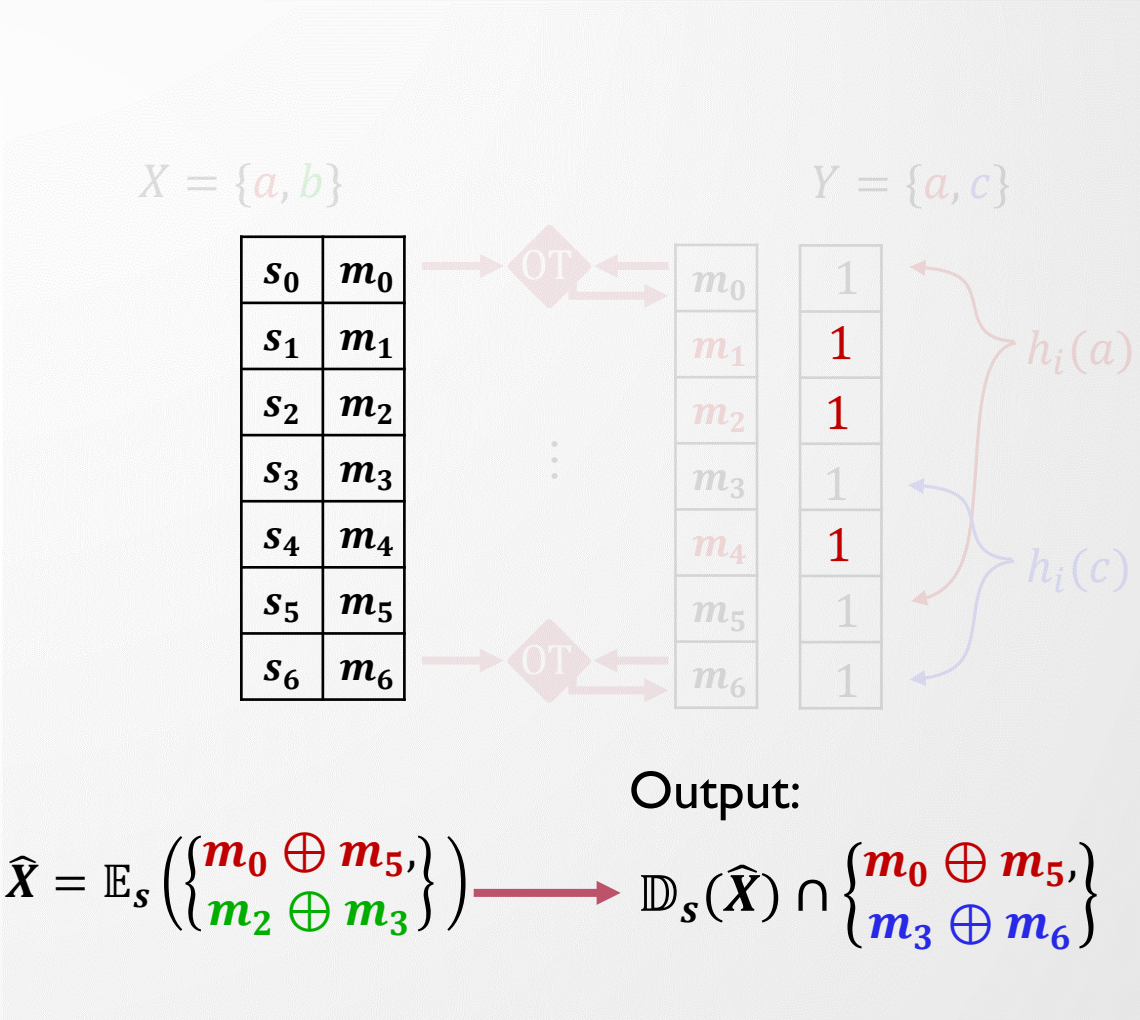
0,1 $\kappa$ 0,1 0,1 0,1 0,1 $\kappa$ $\kappa\kappa$ 0,1 $\kappa$

1 2 $2$ $1$ $2$ $mm$ ones

Goal – restrict the Receiver to a valid Bloom filter

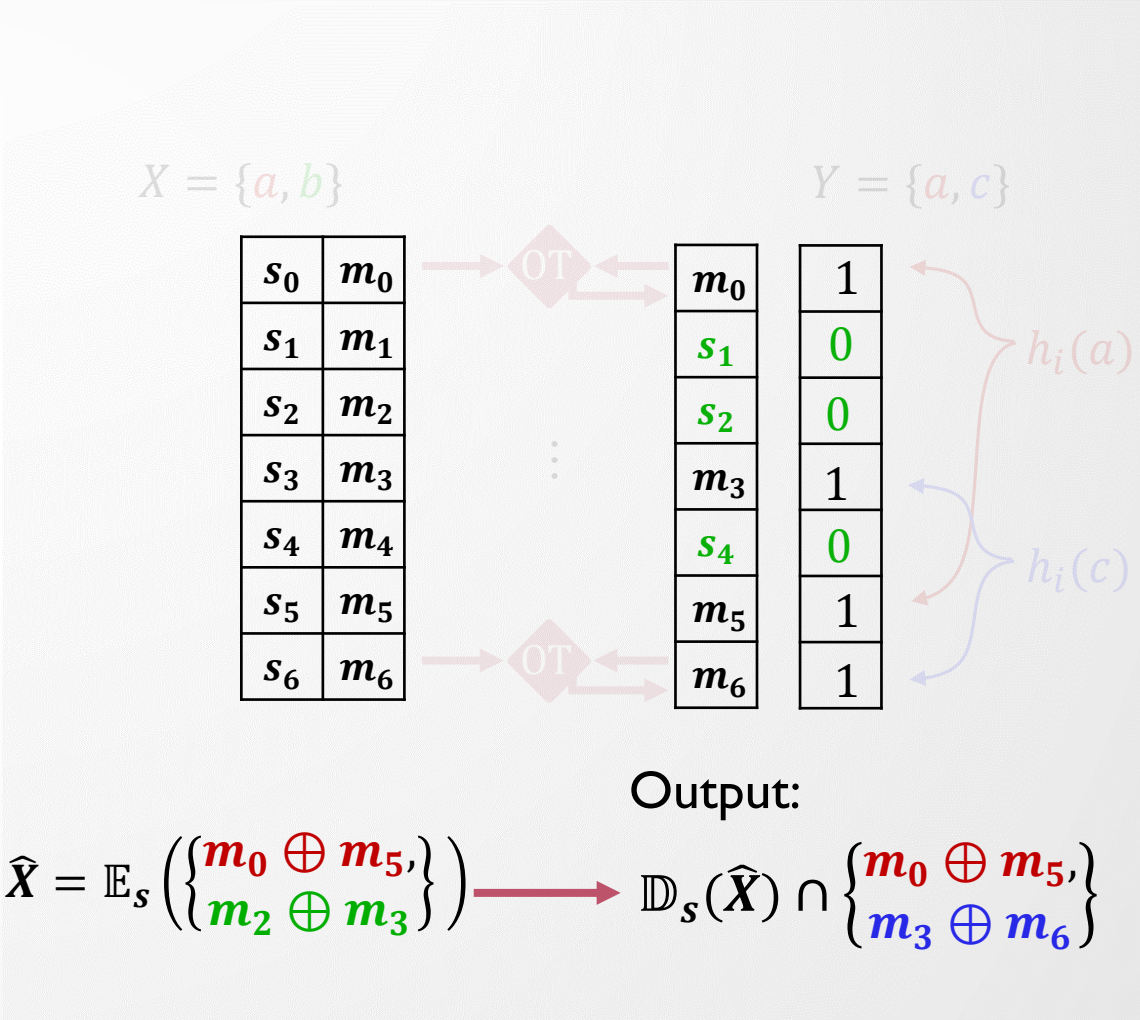
- Bloom filter of $m$ bits contains 1 2 $m$ ones

- Make Receiver prove zero choice bits
  - $s$ 1 1 1 , … , $s_m$
  - Encrypt summary values with $s$Generate a $\frac{m}{2}$ out of $m$ secret sharing of $s$
    - $s_1, \dots, s_m$
  - Transmit $s_i$ as the $i$th zero OT message
  - Encrypt summary values with $s$



# 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 $s$
    - $s_1, \ldots, s_m$
  - Transmit $s_i$ as the $i$th zero OT message
  - Encrypt summary values with $s$



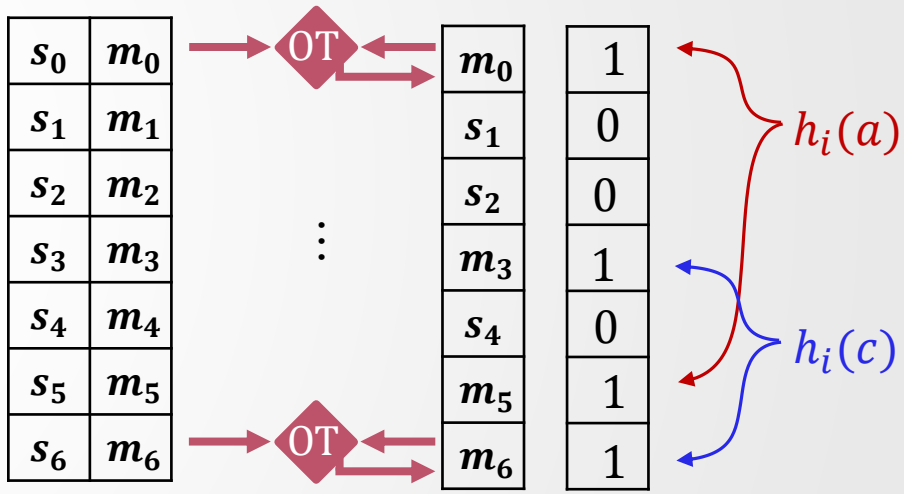
# Warm-Up: The DongChenWen13 Approach

[RindalRosulek17, Lambaek17]

Is this secure?

- $\frac{m}{2}$ ones
- Selective failure attack by the Sender…

$X = \{a, b\}$ $\qquad$ $Y = \{a, c\}$




Output:

$$\widehat{X} = \mathbb{E}_s\left(\begin{Bmatrix} m_0 \oplus m_5, \\ m_2 \oplus m_3 \end{Bmatrix}\right) \longrightarrow \mathbb{D}_s(\widehat{X}) \cap \begin{Bmatrix} m_0 \oplus m_5, \\ m_3 \oplus m_6 \end{Bmatrix}$$

# Warm-Up: The DongChenWen13 Approach

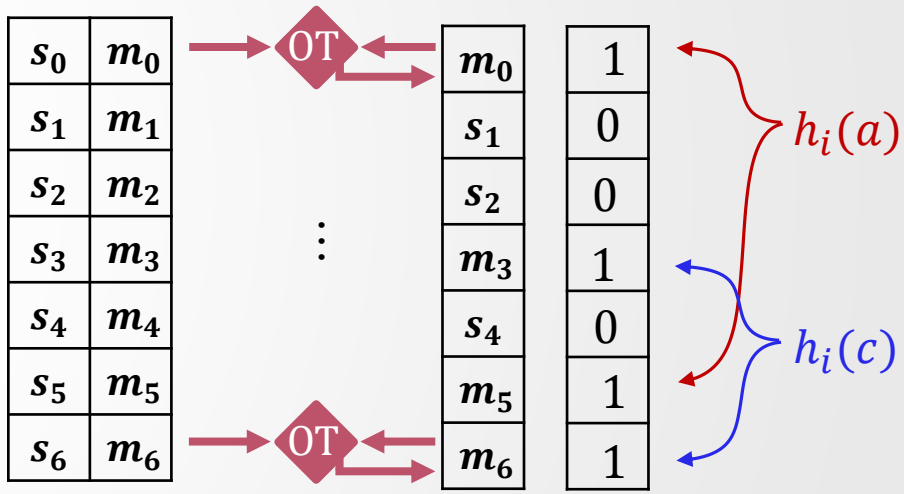
[RindalRosulek17, Lambaek17]

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

Is this secure?

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

$X = \{a, b\}$ $\qquad$ $Y = \{a, c\}$




Output:

$$\hat{X} = \mathbb{E}_s\left(\begin{Bmatrix} m_0 \oplus m_5, \\ m_2 \oplus m_3 \end{Bmatrix}\right) \longrightarrow \mathbb{D}_s(\hat{X}) \cap \begin{Bmatrix} m_0 \oplus m_5, \\ m_3 \oplus m_6 \end{Bmatrix}$$

# Warm-Up: The DongChenWen13 Approach

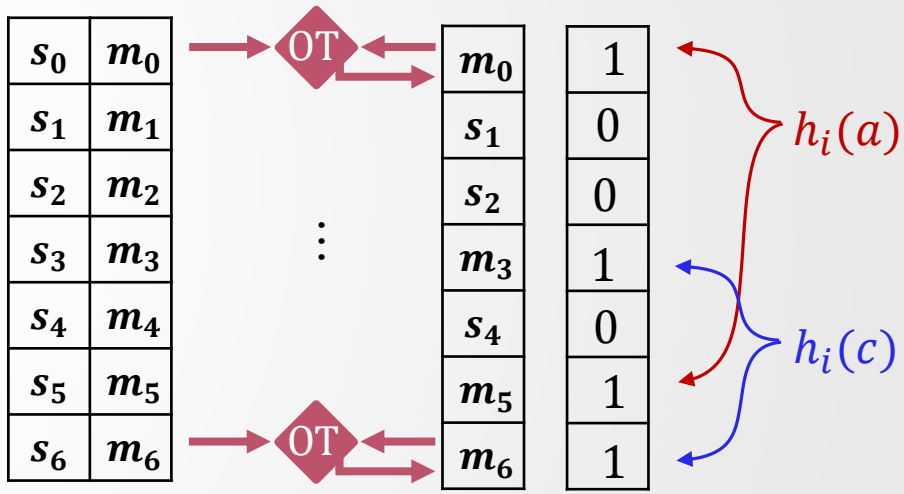
[RindalRosulek17, Lambaek17]

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

Is this secure?

- Selective failure attack by the Sender…
  Selective failure attack by the Sender…

$X = \{a, b\}$ $\qquad\qquad$ $Y = \{a, c\}$



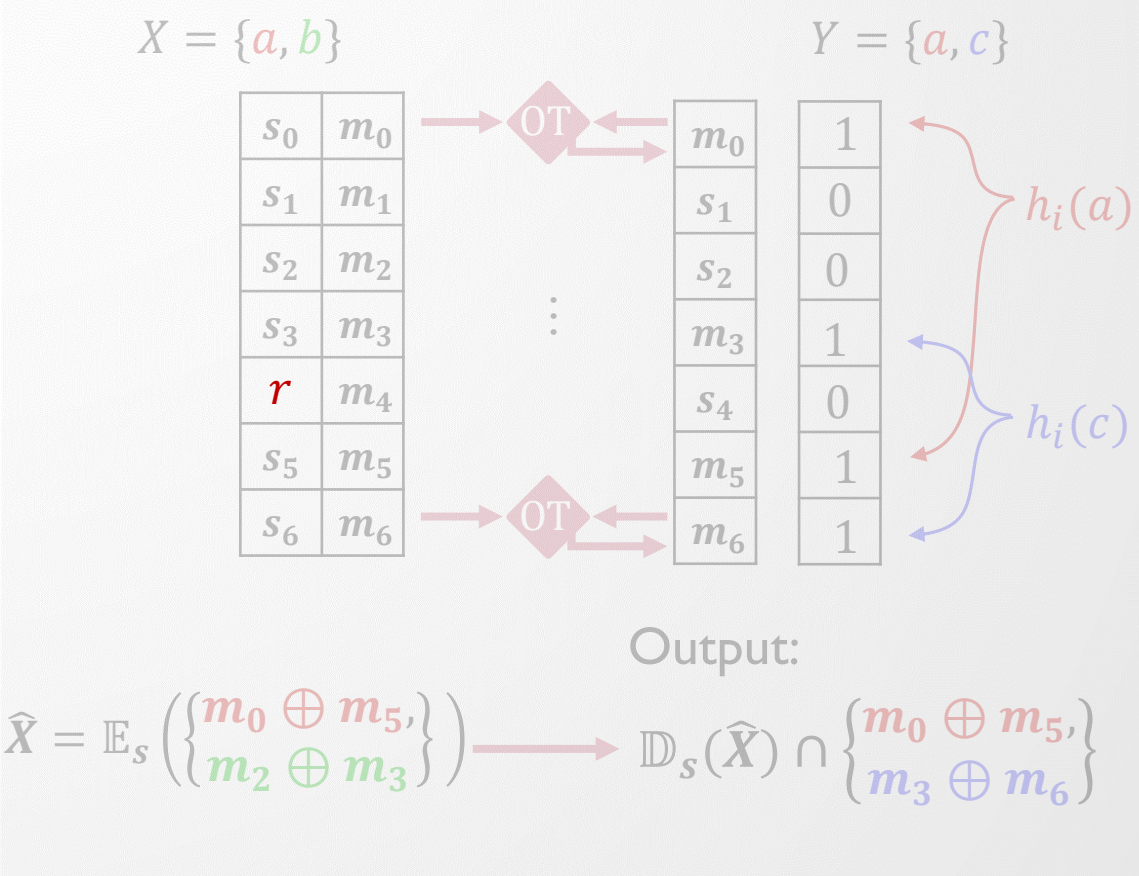

Output:

$$\hat{X} = \mathbb{E}_s\left(\begin{Bmatrix} m_0 \oplus m_5, \\ m_2 \oplus m_3 \end{Bmatrix}\right) \longrightarrow \mathbb{D}_s(\hat{X}) \cap \begin{Bmatrix} m_0 \oplus m_5, \\ m_3 \oplus m_6 \end{Bmatrix}$$

# Warm-Up: The DongChenWen13 Approach

[RindalRosulek17, Lambaek17]

- 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 $r$



# Warm-Up: The DongChenWen13 Approach

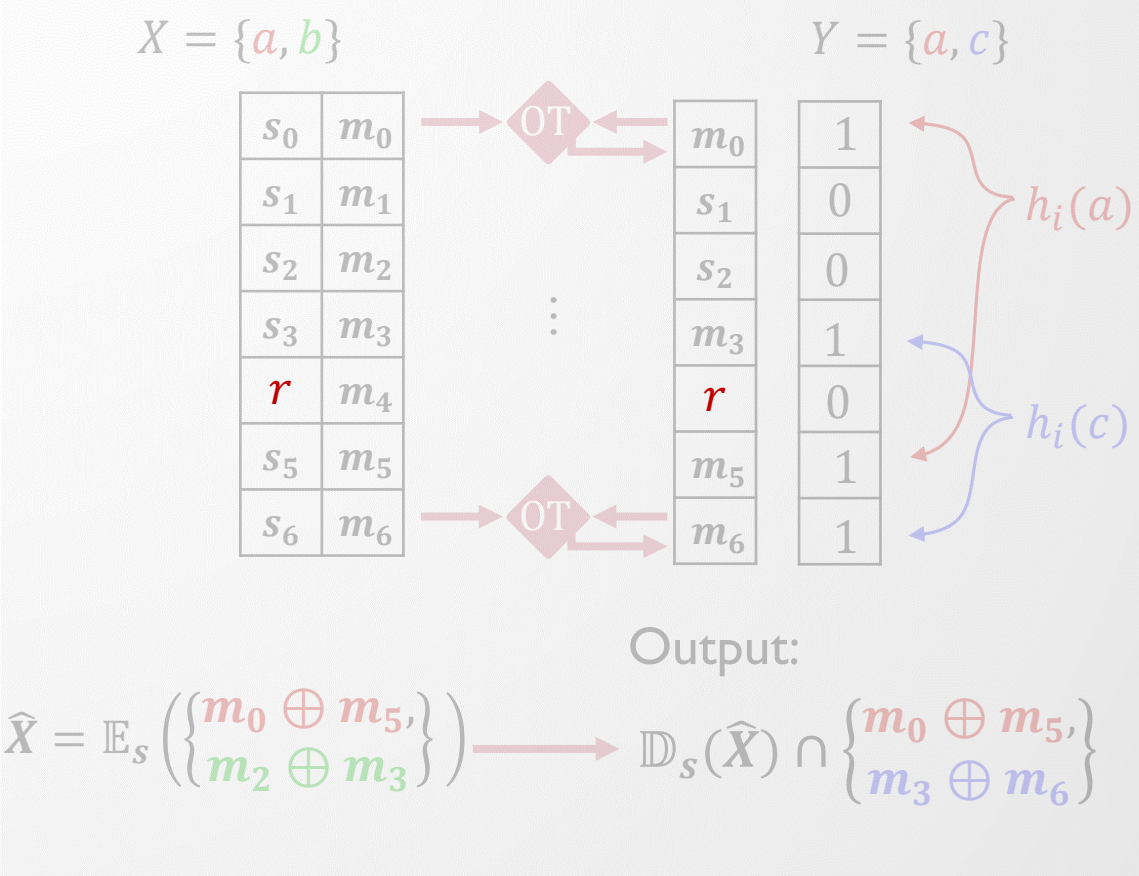
[RindalRosulek17, Lambaek17]

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

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



# Warm-Up: The DongChenWen13 Approach

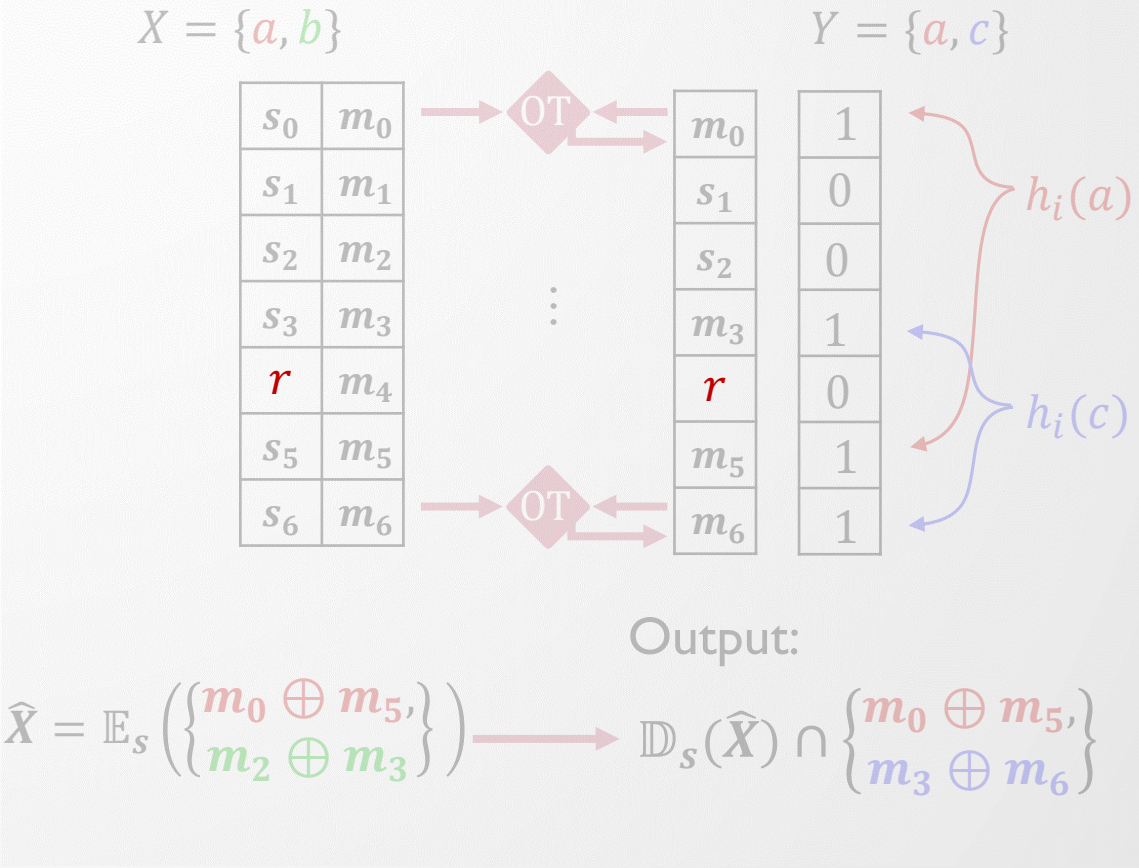
[RindalRosulek17, Lambaek17]

$\forall yy \in YY \quad : \quad h\ i\ h\ h\ i\ ii\ h\ i\ y\ yy\ y \neq 4$

*Is* this secure?

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

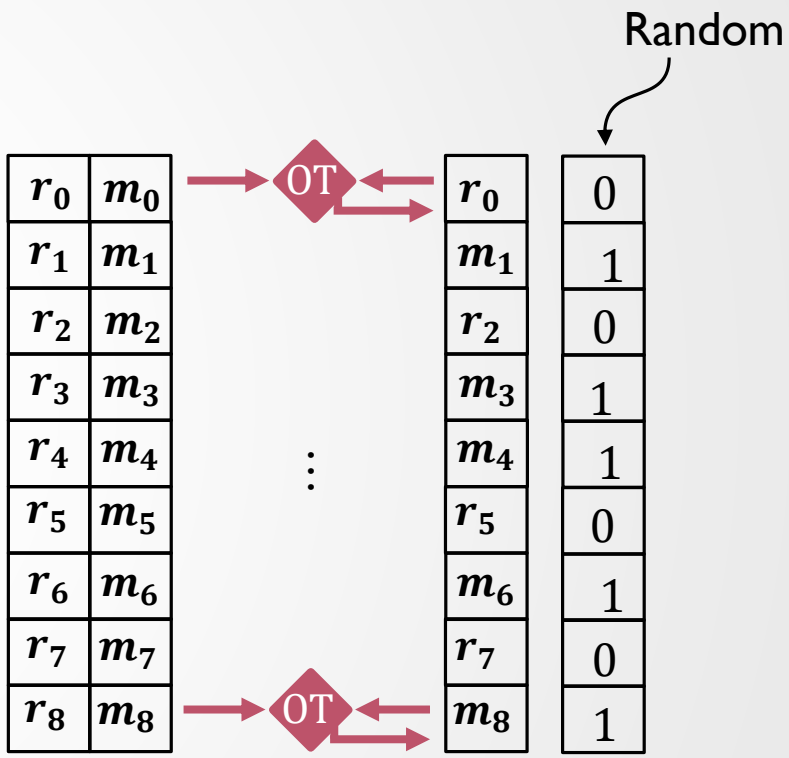
- Example Attack:
  - replace $s_4$ with random value $r$
  - Can not reconstruct $s$ if $r$ is picked up
  - Can not be simulated! Can not be simulated!



# Cut and Choose Approach

[RindalRosulek17]

Make Receiver p

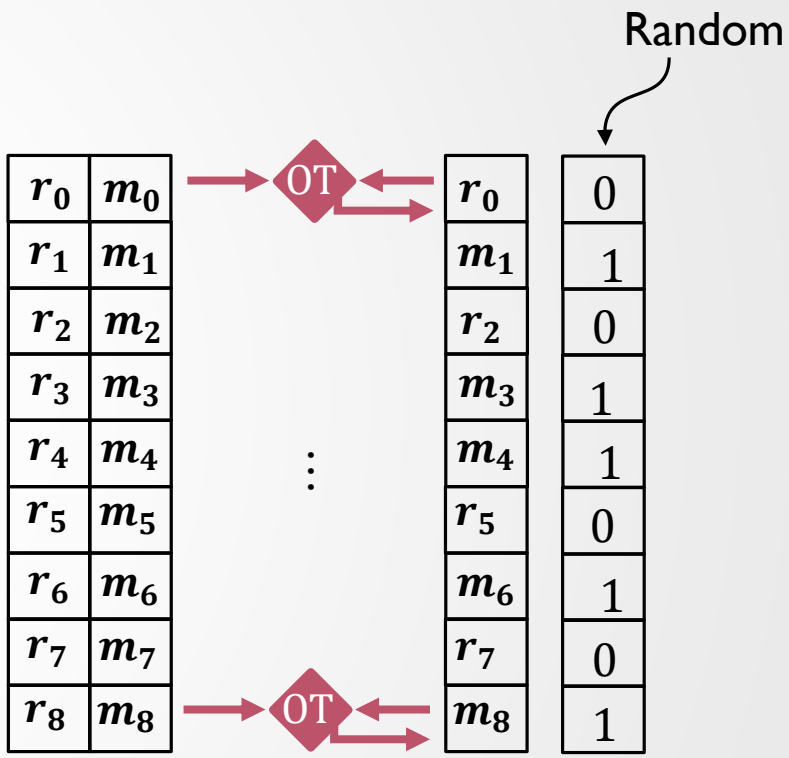


# Cut and Choose Approach

[RindalRosulek17]

Make Receiver p

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

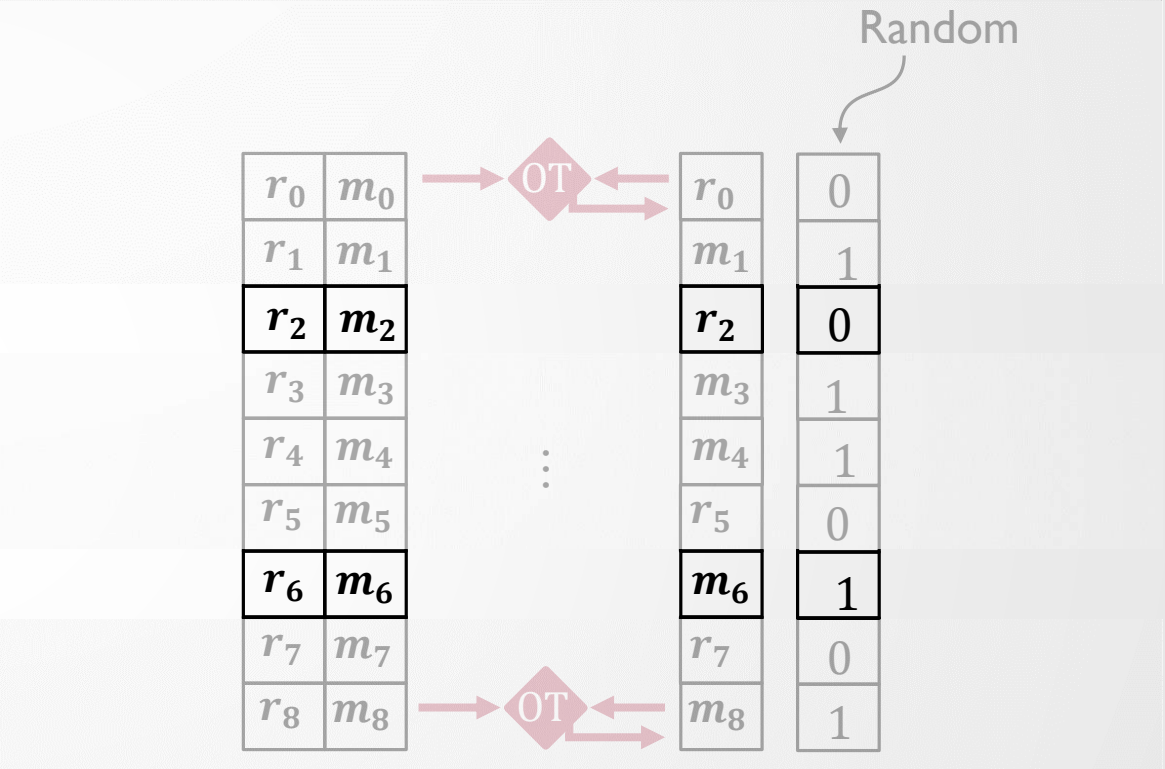


# Cut and Choose Approach

[RindalRosulek17]

Make Receiver p

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

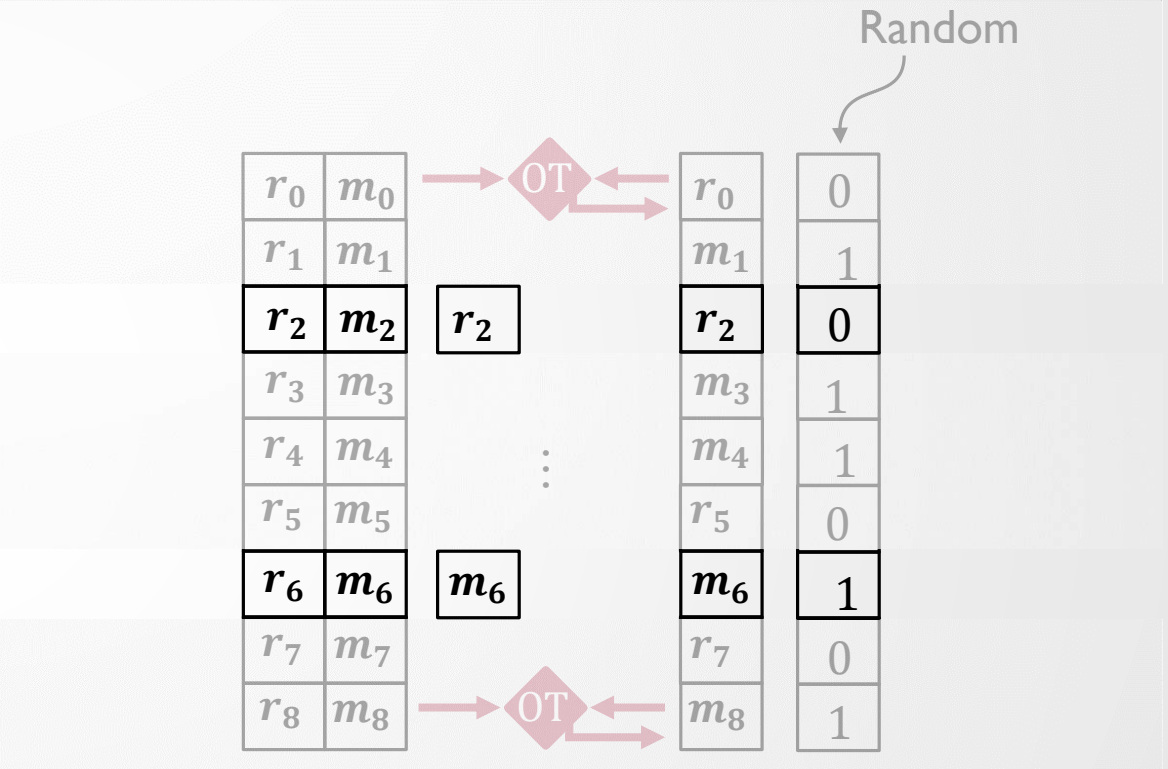


# Cut and Choose Approach

[RindalRosulek17]

Make Receiver p 2 zero bits, aborts otherwise

- Sender challenges on a subset of OT
  - Receiver must reveal select bits
  - Expect to see 1 2 zero bits, aborts otherwise

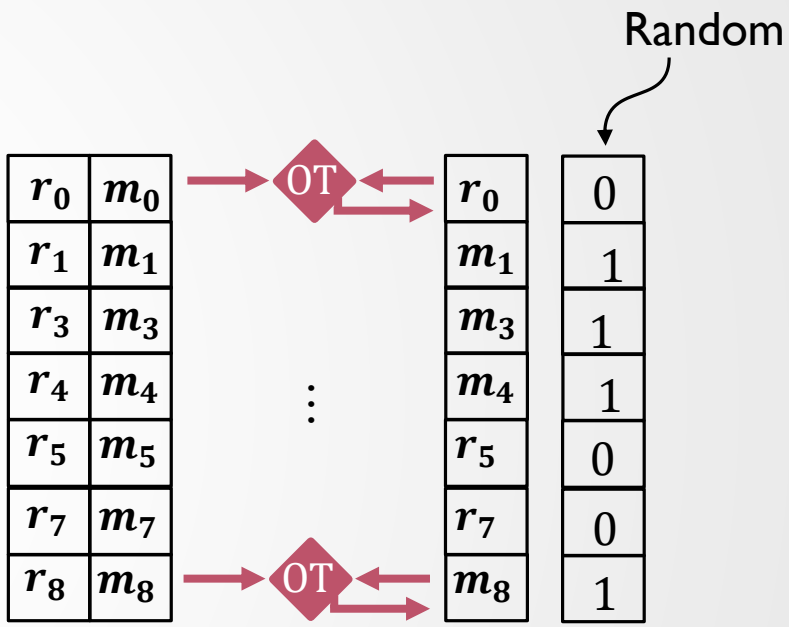


# Cut and Choose Approach

[RindalRosulek17]

- Make Receiver prove zero bits in an input-independent way

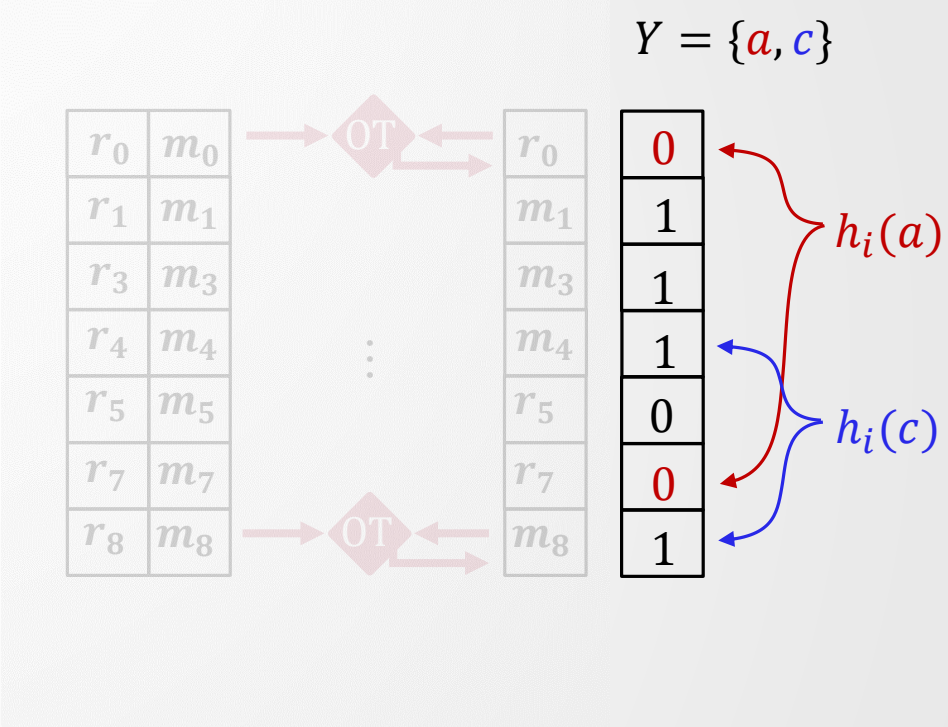
- Receiver uses *random* OT select bits
- Sender challenges on a subset of OT
  - Receiver must reveal select bits
  - Expect to see $\frac{1}{2}$ zero bits, aborts otherwise



# Cut and Choose Approach

[RindalRosulek17]

- Issue: Remaining OTs do not form valid Bloom filter

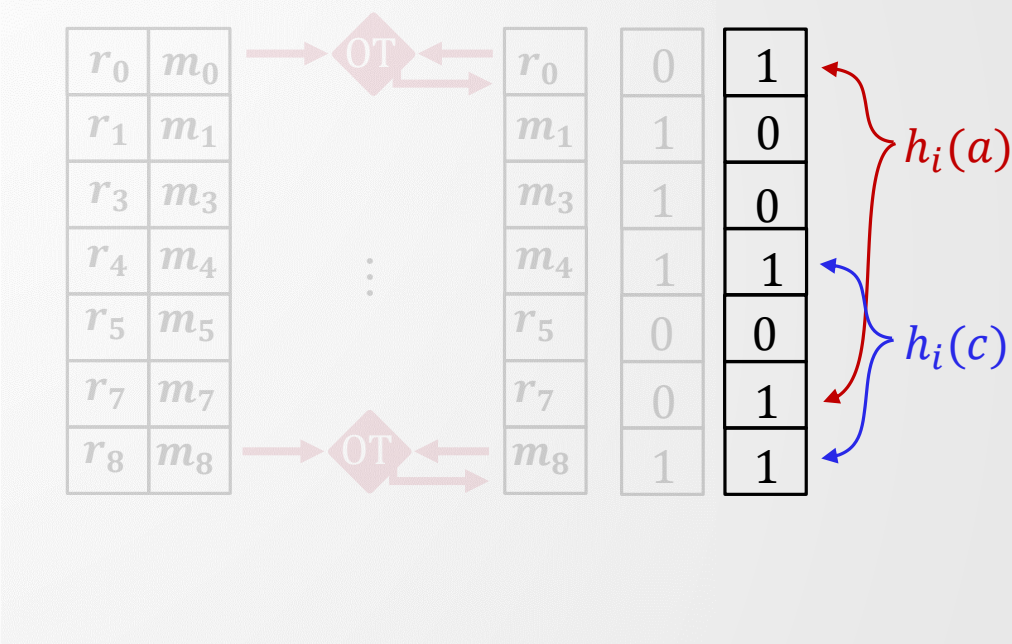


# Cut and Choose Approach

[RindalRosulek17]

1

$Y = \{a, c\}$

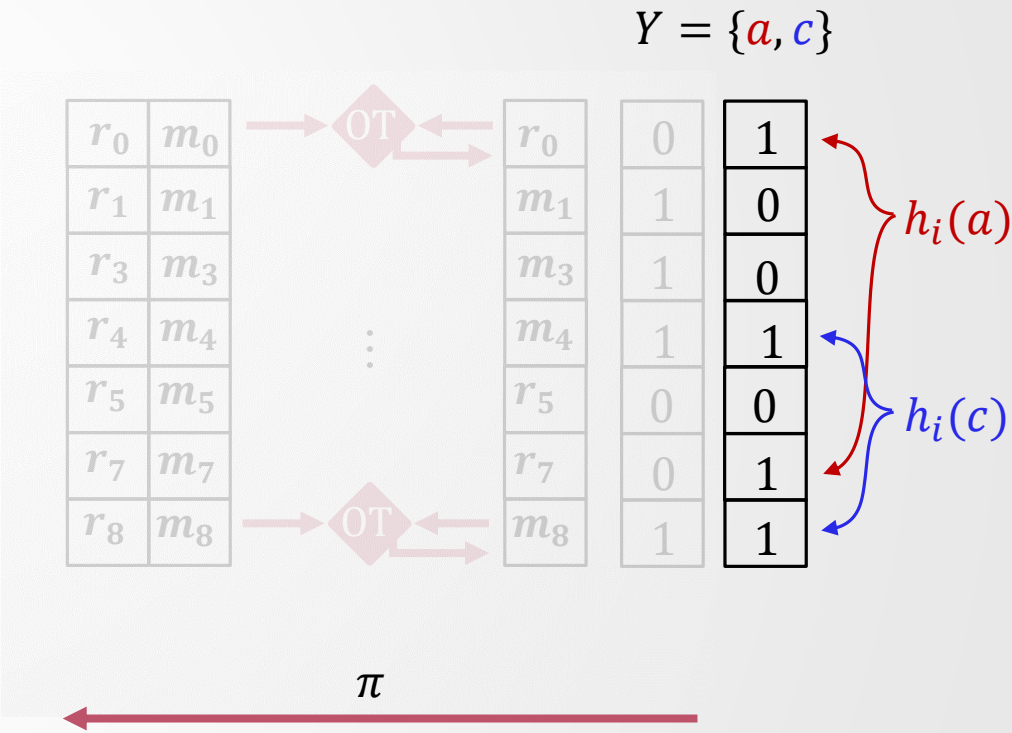


# Cut and Choose Approach

[RindalRosulek17]

l andom OTs →desired $BF$

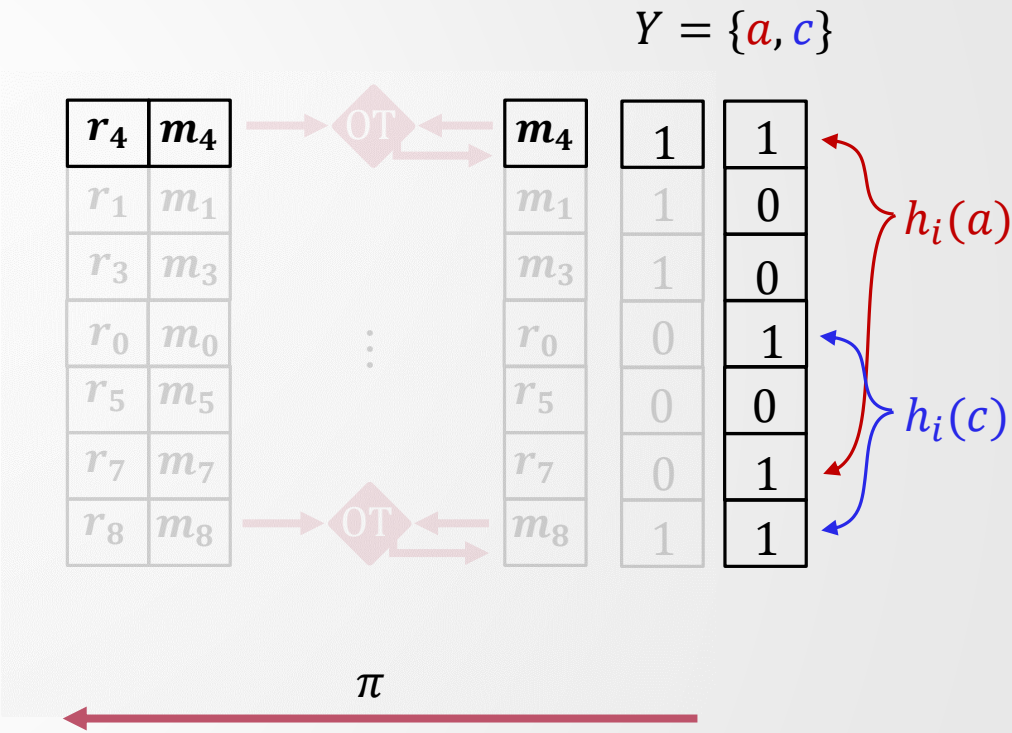
- Randomly permute OTs to form Bloom filter
  - $\pi$ random OTs → desired $BF$



# Cut and Choose Approach

[RindalRosulek17]

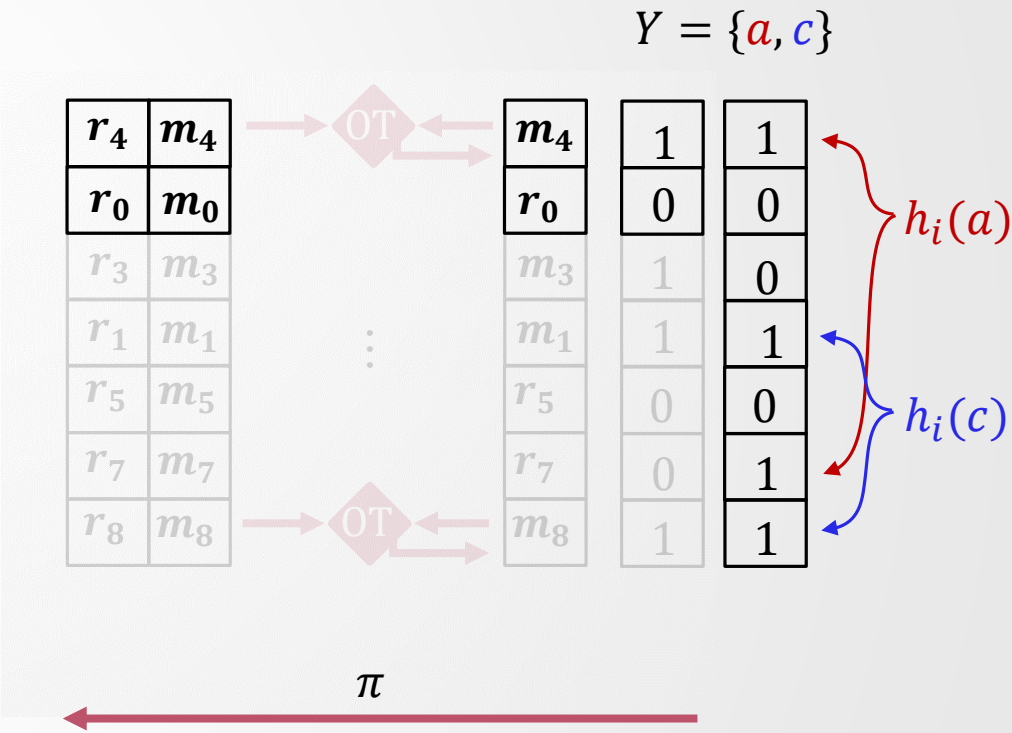
- Issue: Remaining OTs do not form valid Bloom filter
- Constructs desired Bloom filter
- Randomly permute OTs to form Bloom filter
  - $\pi(\text{random OTs}) \rightarrow \text{desired } BF$



# Cut and Choose Approach

[RindalRosulek17]

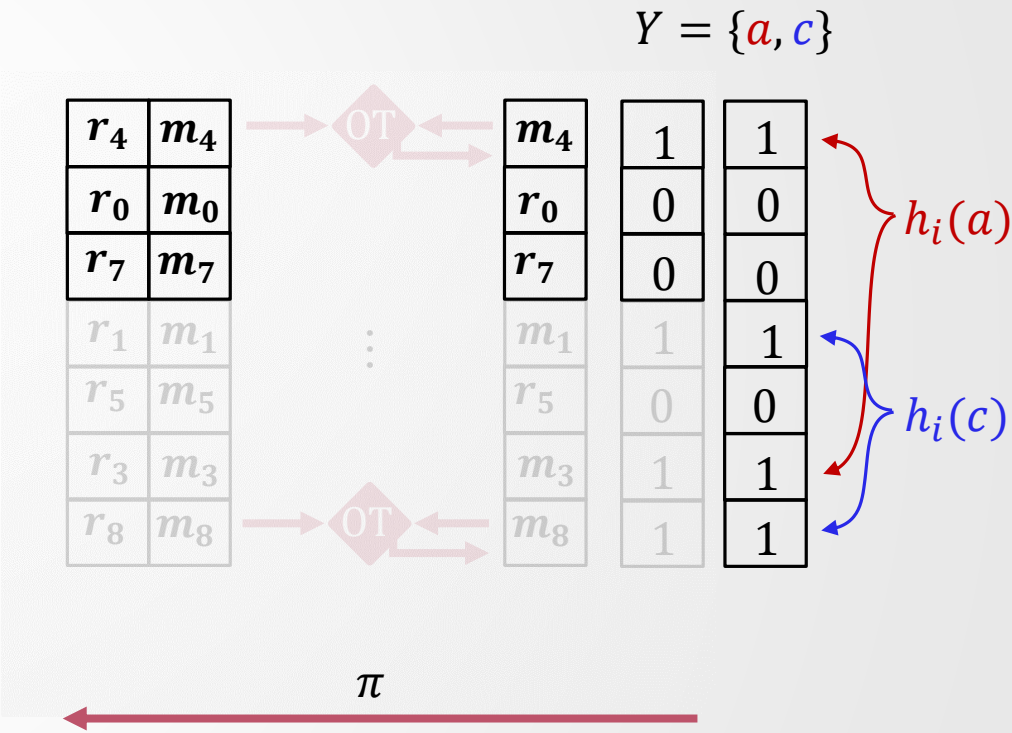
- Issue: Remaining OTs do not form valid Bloom filter
- Constructs desired Bloom filter
- Randomly permute OTs to form Bloom filter
  - $\pi(\text{random OTs}) \rightarrow \text{desired } BF$



# Cut and Choose Approach

[RindalRosulek17]

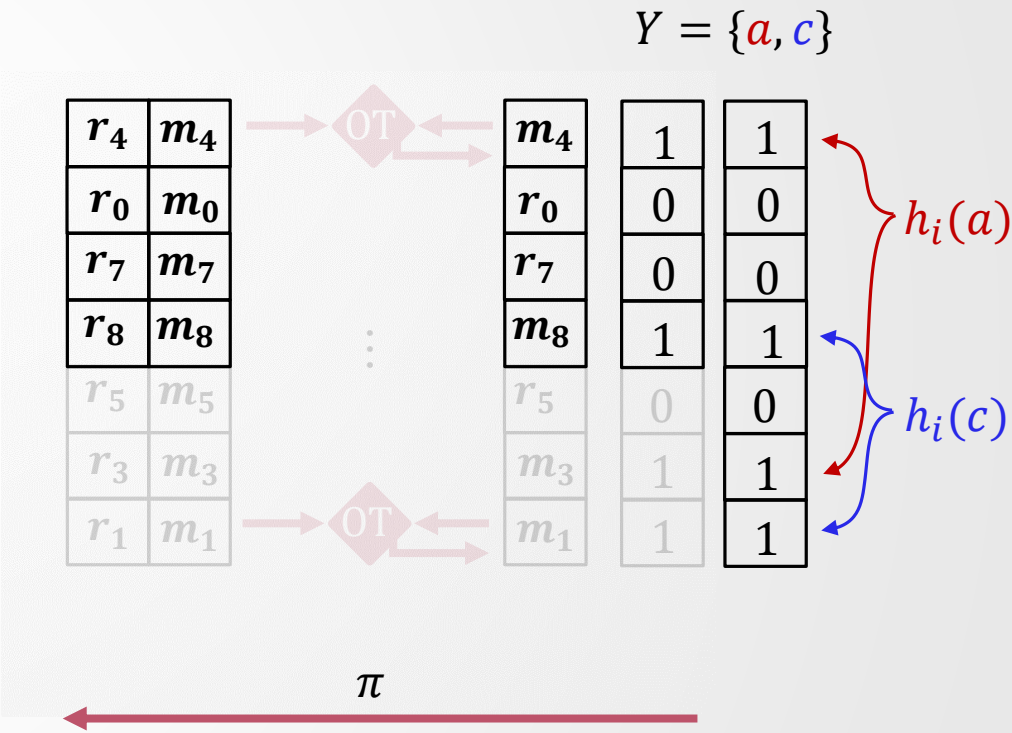
- Issue: Remaining OTs do not form valid Bloom filter
- Constructs desired Bloom filter
- Randomly permute OTs to form Bloom filter
  - $\pi(\text{random OTs}) \rightarrow \text{desired } BF$



# Cut and Choose Approach

[RindalRosulek17]

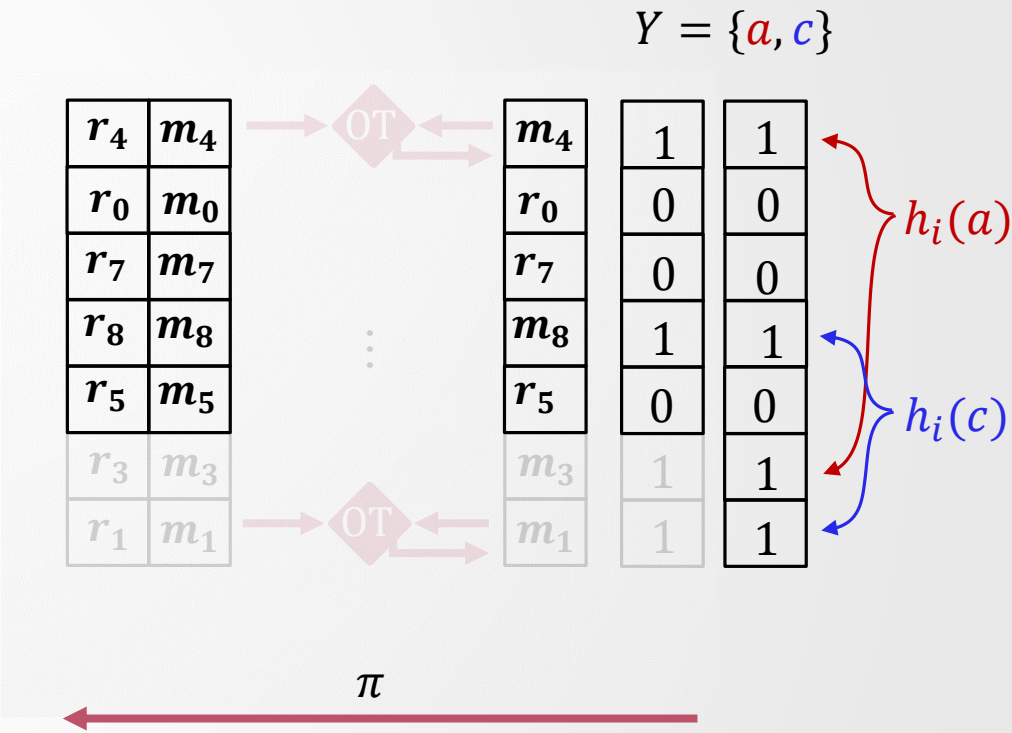
- Issue: Remaining OTs do not form valid Bloom filter
- Constructs desired Bloom filter
- Randomly permute OTs to form Bloom filter
  - $\pi(\text{random OTs}) \rightarrow \text{desired } BF$



# Cut and Choose Approach

[RindalRosulek17]

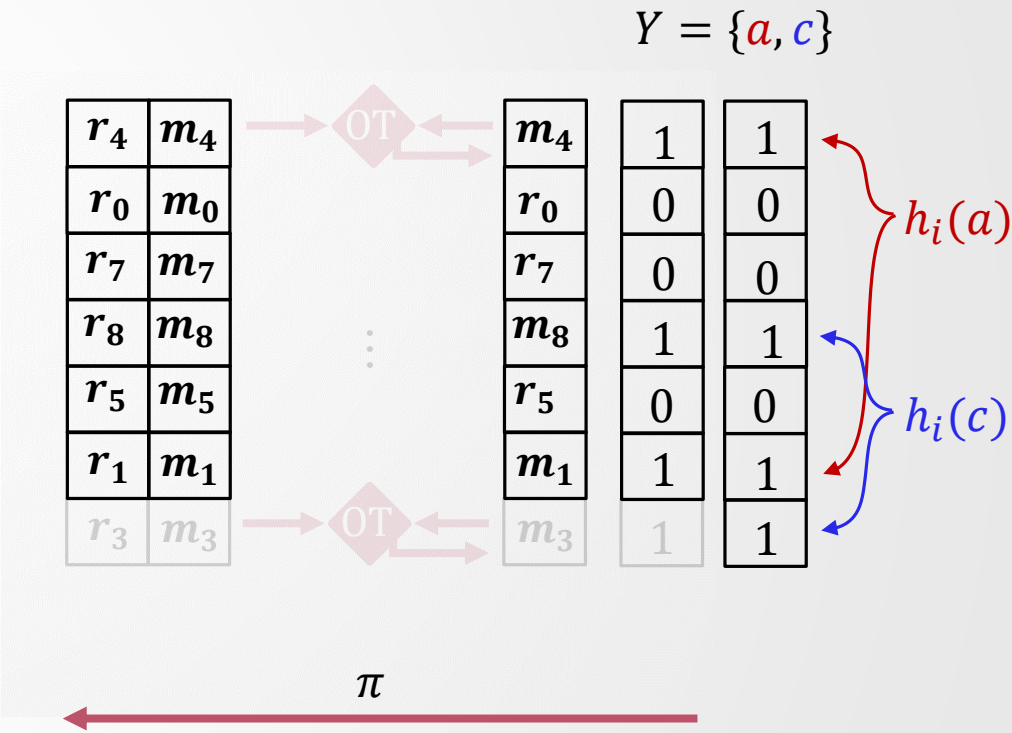
- Issue: Remaining OTs do not form valid Bloom filter
- Constructs desired Bloom filter
- Randomly permute OTs to form Bloom filter
  - $\pi(\text{random OTs}) \rightarrow \text{desired } BF$



# Cut and Choose Approach

[RindalRosulek17]

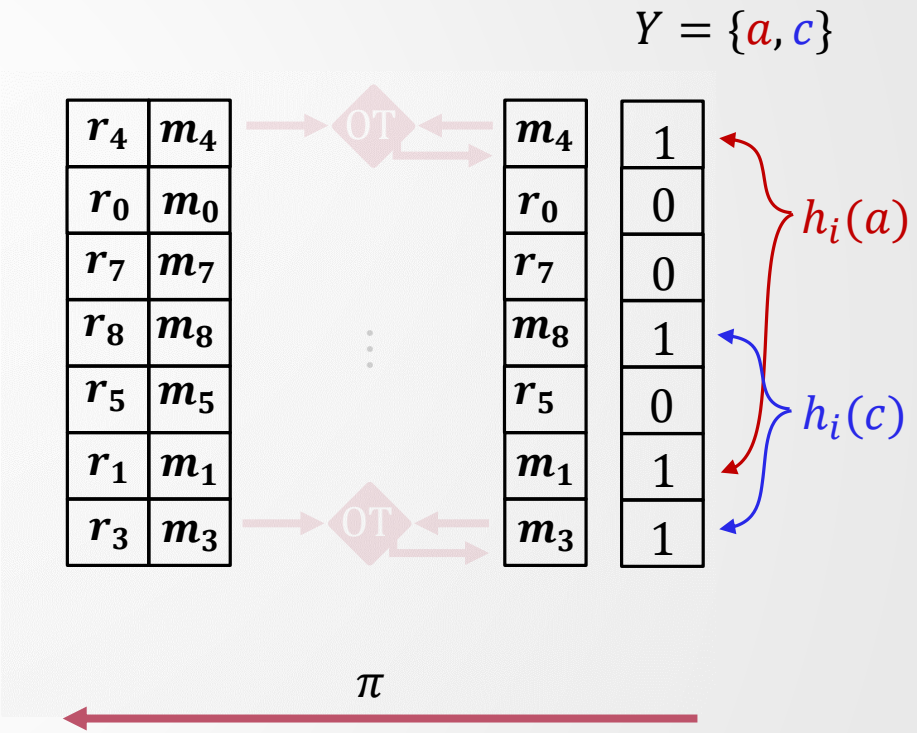
- Issue: Remaining OTs do not form valid Bloom filter
- Constructs desired Bloom filter
- Randomly permute OTs to form Bloom filter
  - $\pi(\text{random OTs}) \rightarrow \text{desired } BF$



# Cut and Choose Approach

[RindalRosulek17]

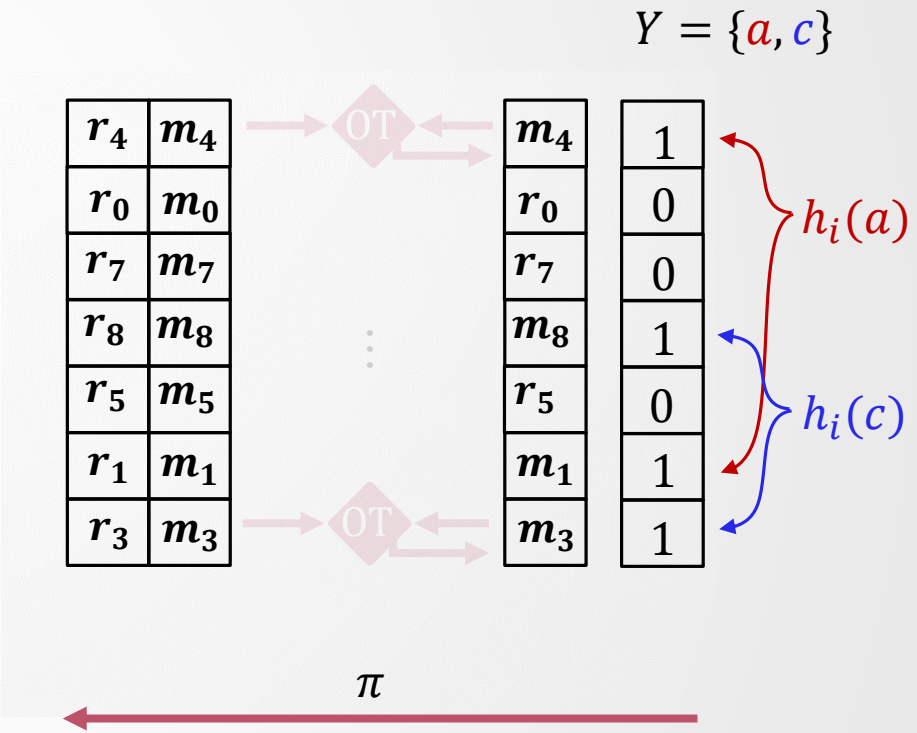
- Issue: Remaining OTs do not form valid Bloom filter
- Constructs desired Bloom filter
- Randomly permute OTs to form Bloom filter
  - $\pi(\text{random OTs}) \rightarrow \text{desired } BF$



# Cut and Choose Approach

[RindalRosulek17]

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



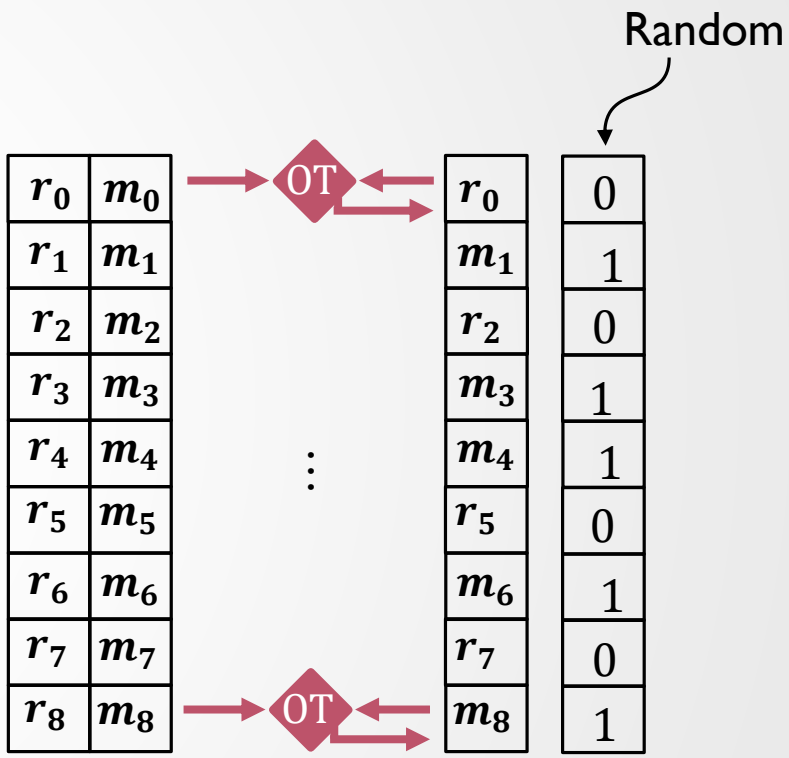

Output:

$$\hat{X} = \begin{Bmatrix} m_4 \oplus m_1, \\ m_2 \oplus m_3 \end{Bmatrix} \longrightarrow \hat{X} \cap \begin{Bmatrix} m_4 \oplus m_1, \\ m_8 \oplus m_3 \end{Bmatrix}$$

# Cut and Choose Parameters

[RindalRosulek17]

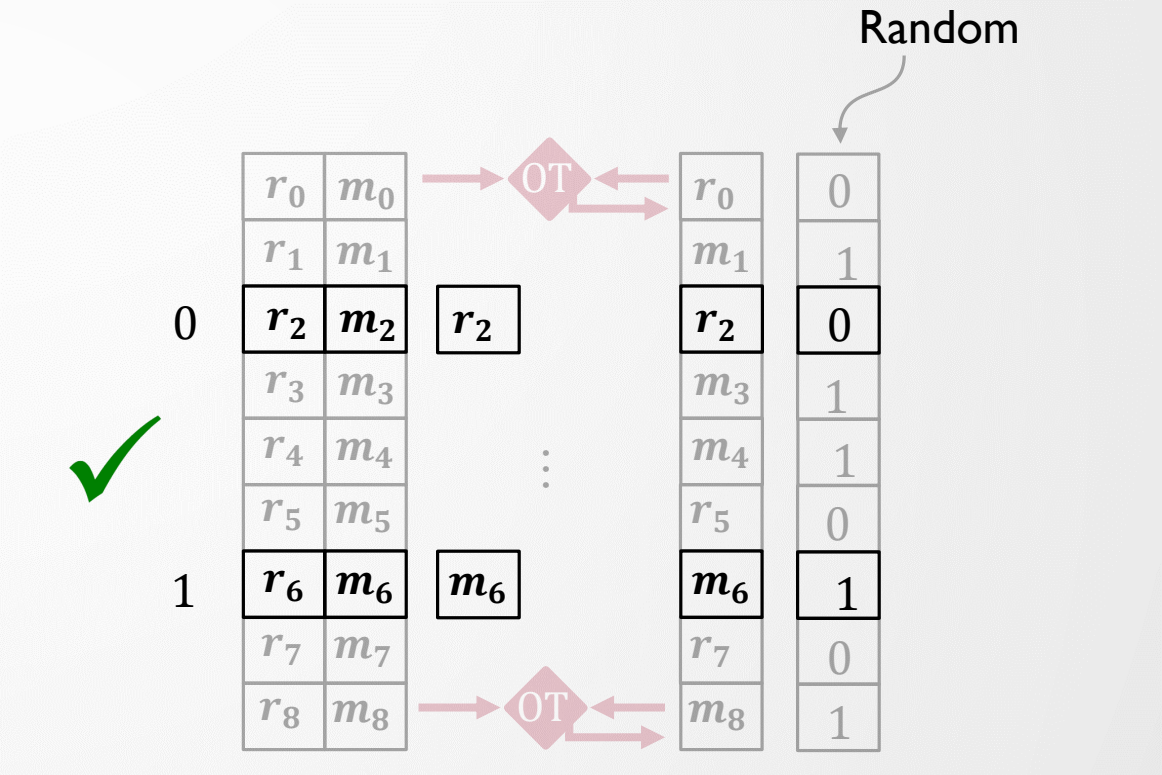
- Issue: Random OTs/Cut-and-Choose may not result in exactly $\frac{1}{2}$ zero select bits!



# Cut and Choose Parameters

[RindalRosulek17]

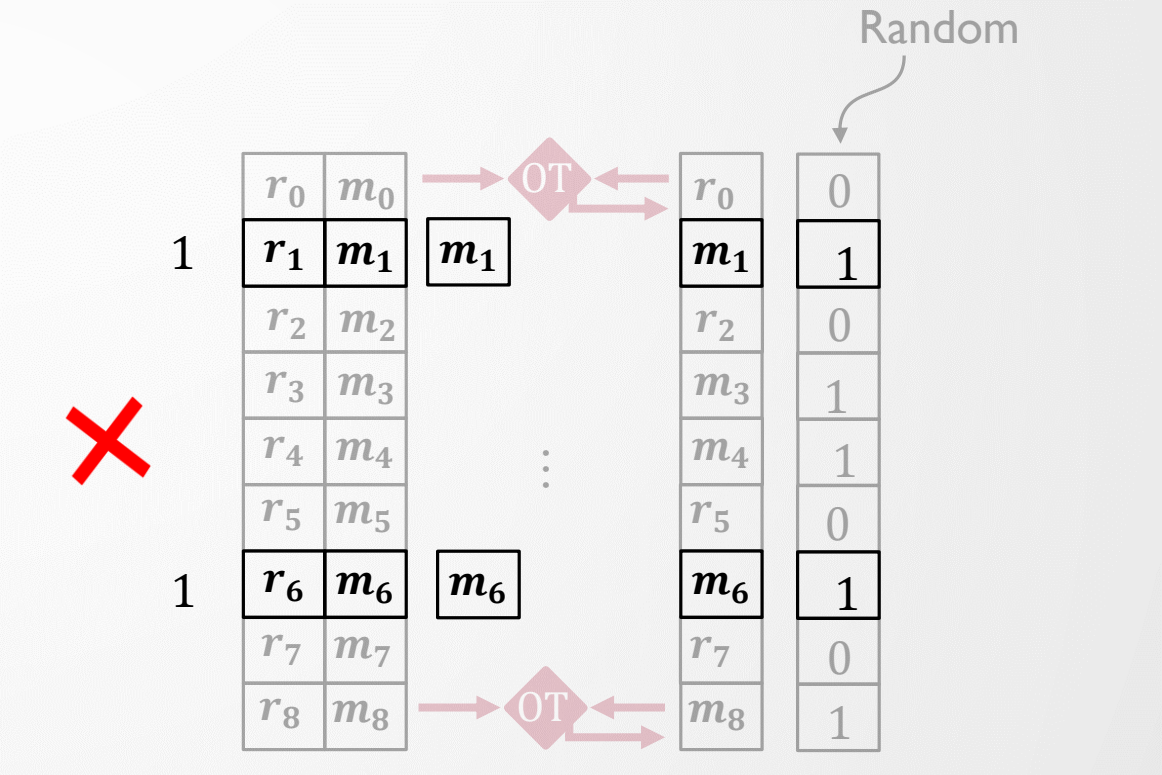
- Issue: Random OTs/Cut-and-Choose may not result in exactly $\frac{1}{2}$ zero select bits!



# Cut and Choose Parameters

[RindalRosulek17]

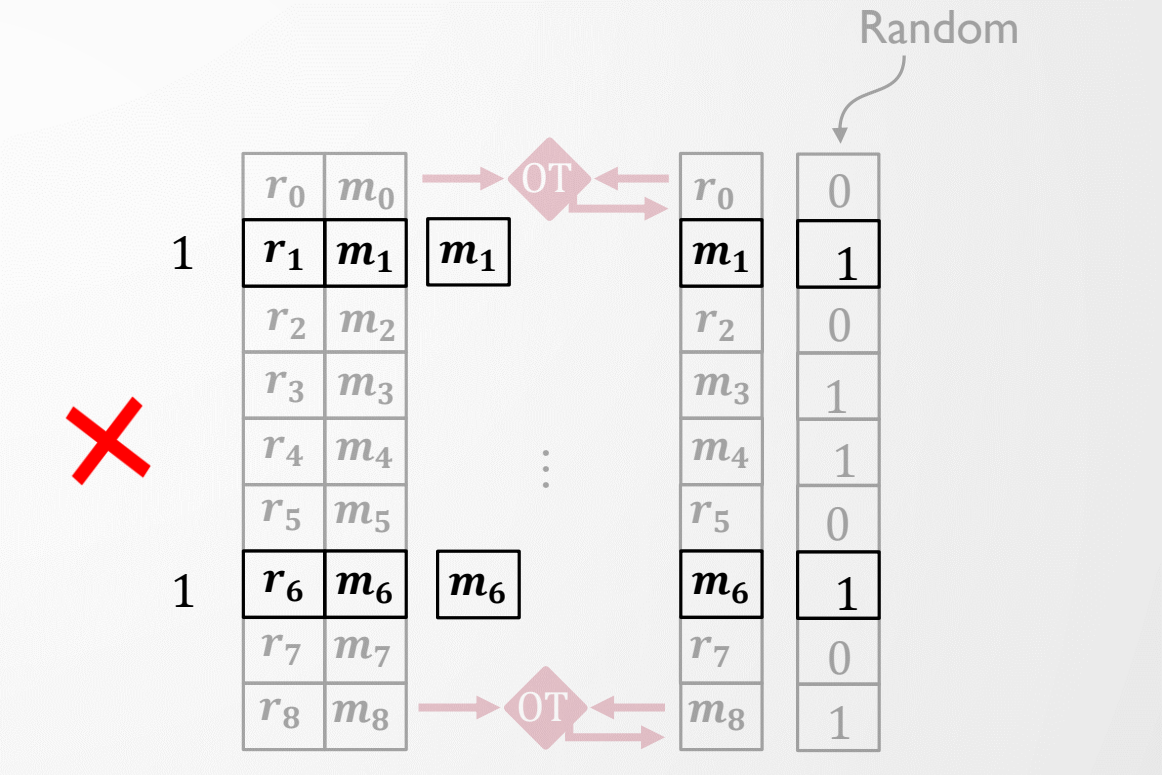
- Issue: Random OTs/Cut-and-Choose may not result in exactly $\frac{1}{2}$ zero select bits!



# Cut and Choose Parameters

[RindalRosulek17]

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

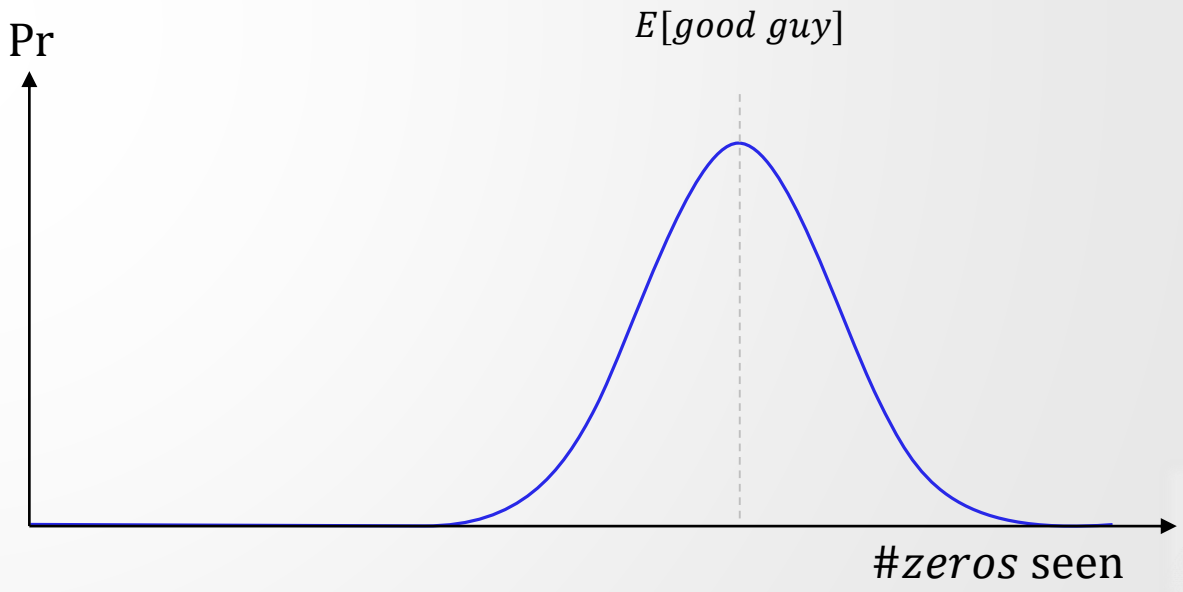


# Cut and Choose Parameters

[RindalRosulek17]

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:
  - $\Pr[good\ guy\ accused] \leq neg(\kappa)$
  -
- Sufficient to check $1\%$ of the OTs!

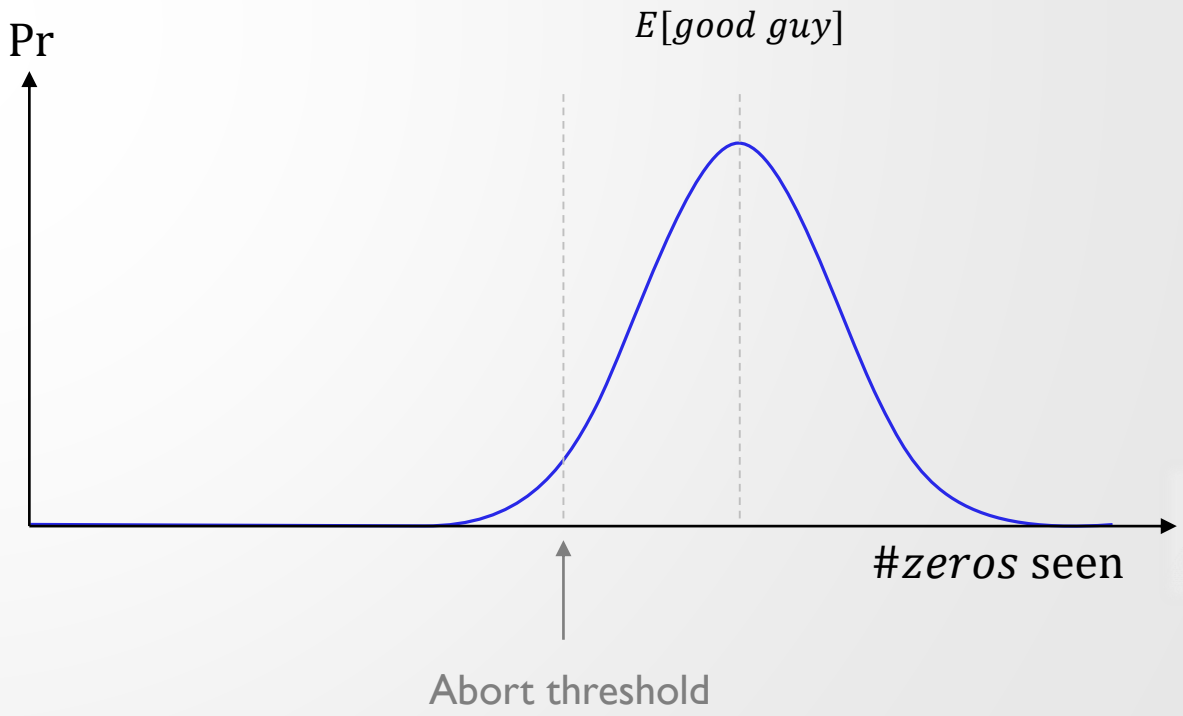


# Cut and Choose Parameters

[RindalRosulek17]

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:
  - $\Pr[good\ guy\ accused] \leq neg(\kappa)$
  -
- Sufficient to check 1% of the OTs!

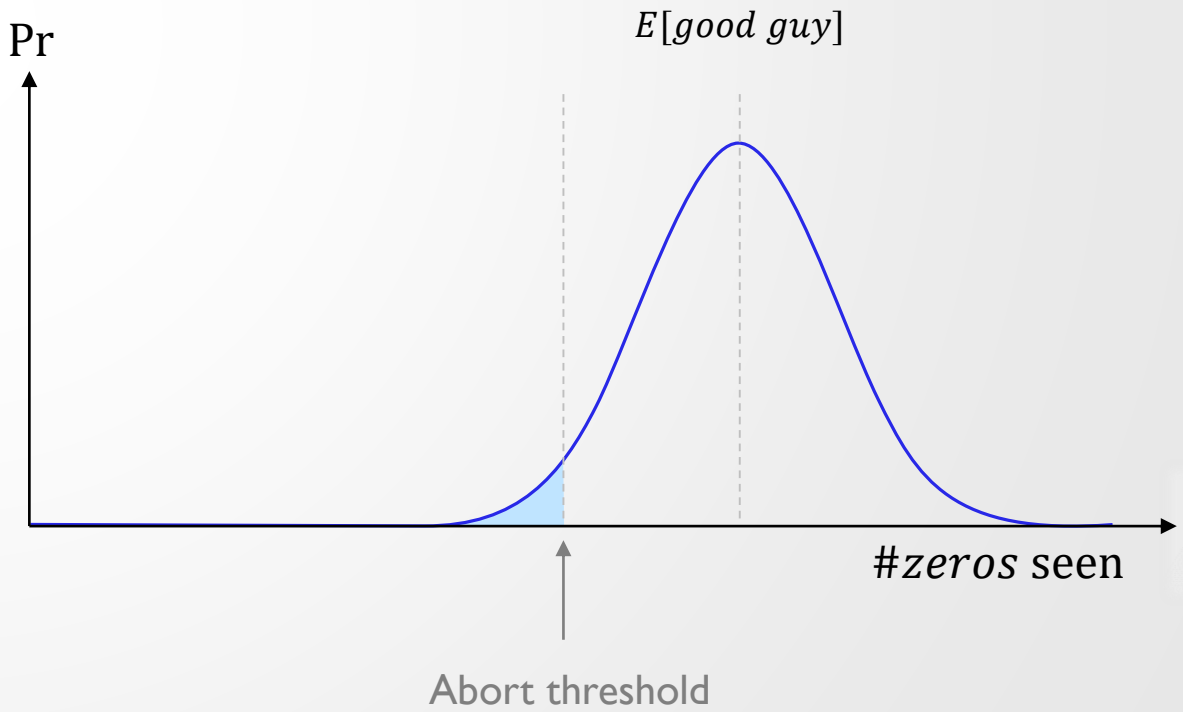


# Cut and Choose Parameters

[RindalRosulek17]

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:
  - $\Pr[good\ guy\ accused] \leq neg(\kappa)$
  -
- Sufficient to check 1% of the OTs!

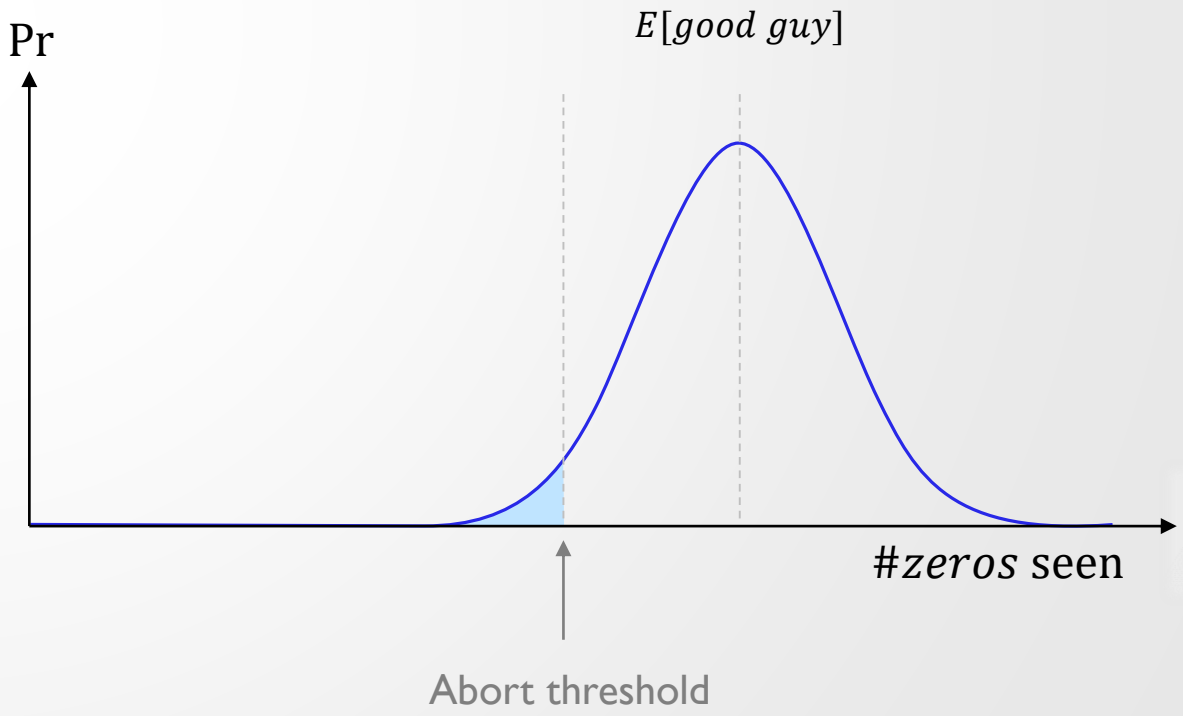


# Cut and Choose Parameters

[RindalRosulek17]

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!



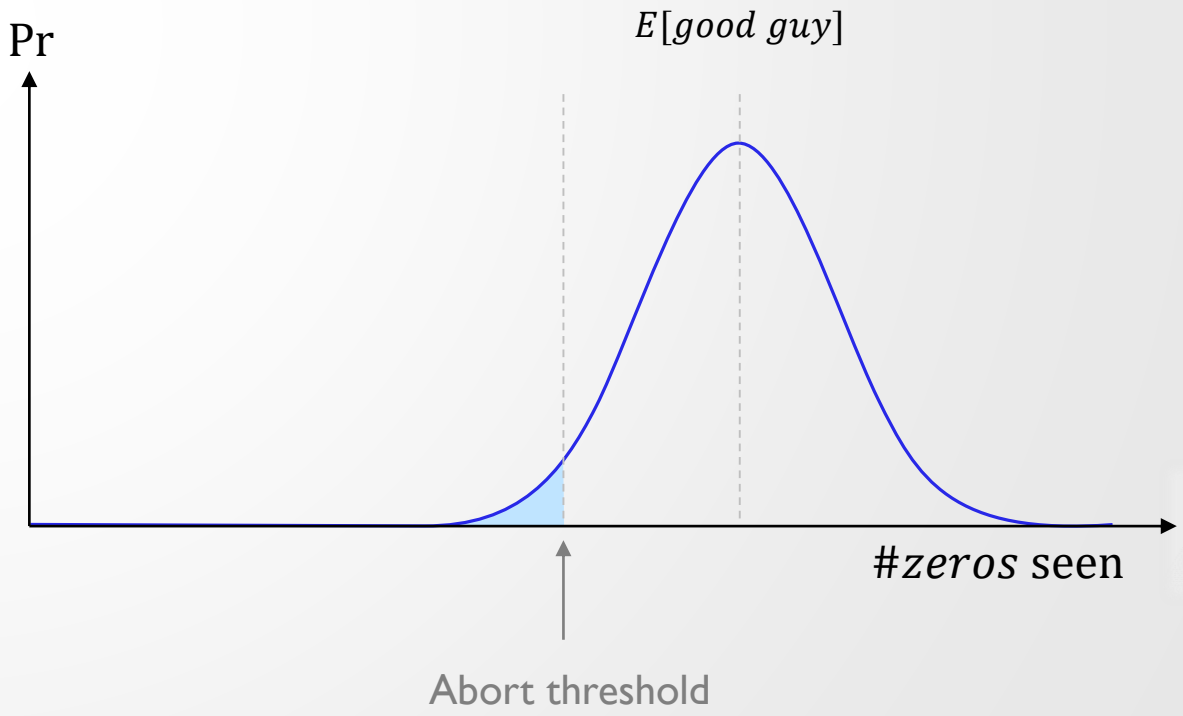
# Cut and Choose Parameters

[RindalRosulek17]

r Pr $Bad\ guy\ not\ caught \leq neg(\kappa)$ $Bad\ guy\ not\ caught$ $BBa$
$add\ gguuyy\ nnoott\ ccaauugghtt\ Bad\ guy\ not\ caught \leq nne$
$egg(\kappa\kappa)$ $Pr\ \ Bad\ guy\ not\ caught \leq neg(\kappa)$

*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:
  - $\Pr[good\ guy\ accused] \leq neg(\kappa)$
  - $Pr\ \ Bad\ guy\ not\ caught \leq neg(\kappa)$
- Use Chernoff Bounds
- Sufficient to check 1% of the OTs!



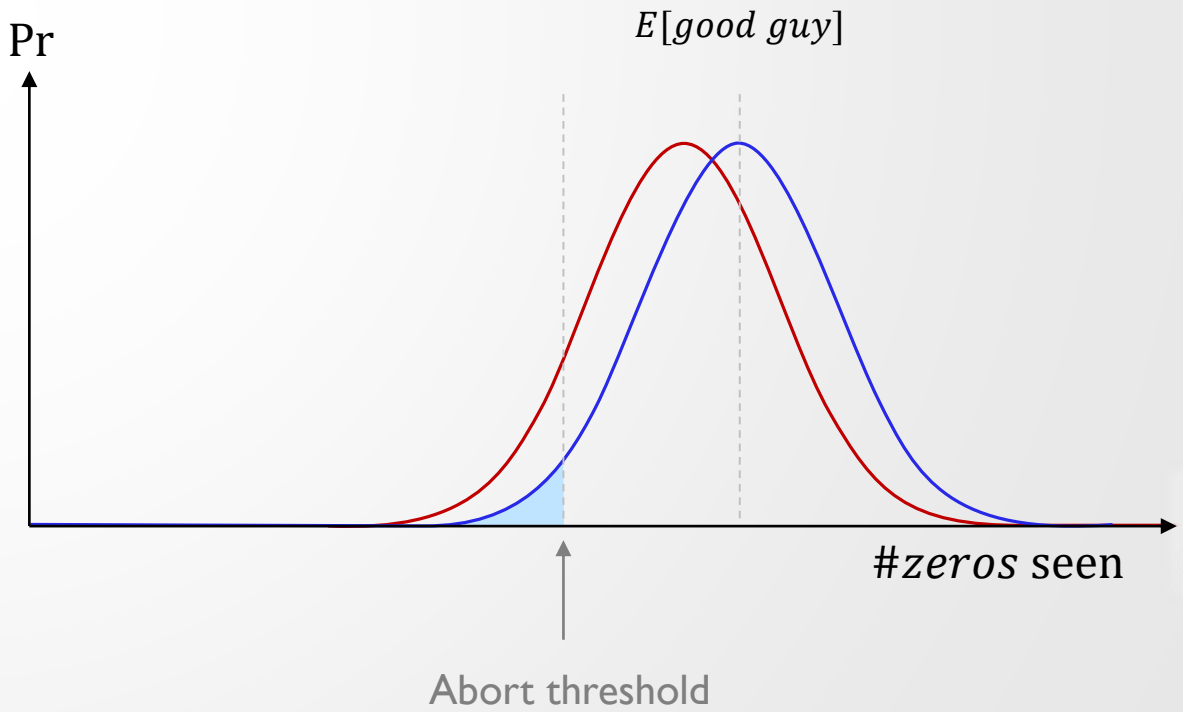
# Cut and Choose Parameters

[RindalRosulek17]

r Pr $Bad\ guy\ not\ caught \leq neg(\kappa)$ $Bad\ guy\ not\ caught\ BBa$
$add\ gguuyy\ nnoott\ ccaauugghtt\ Bad\ guy\ not\ caught \leq nne$
$egg(\kappa\kappa)\ Pr\ \ Bad\ guy\ not\ caught \leq neg(\kappa)$

*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:
  - $\Pr[good\ guy\ accused] \leq neg(\kappa)$
  - $Pr\ \ Bad\ guy\ not\ caught \leq neg(\kappa)$
- Use Chernoff Bounds
- Sufficient to check 1% of the OTs!



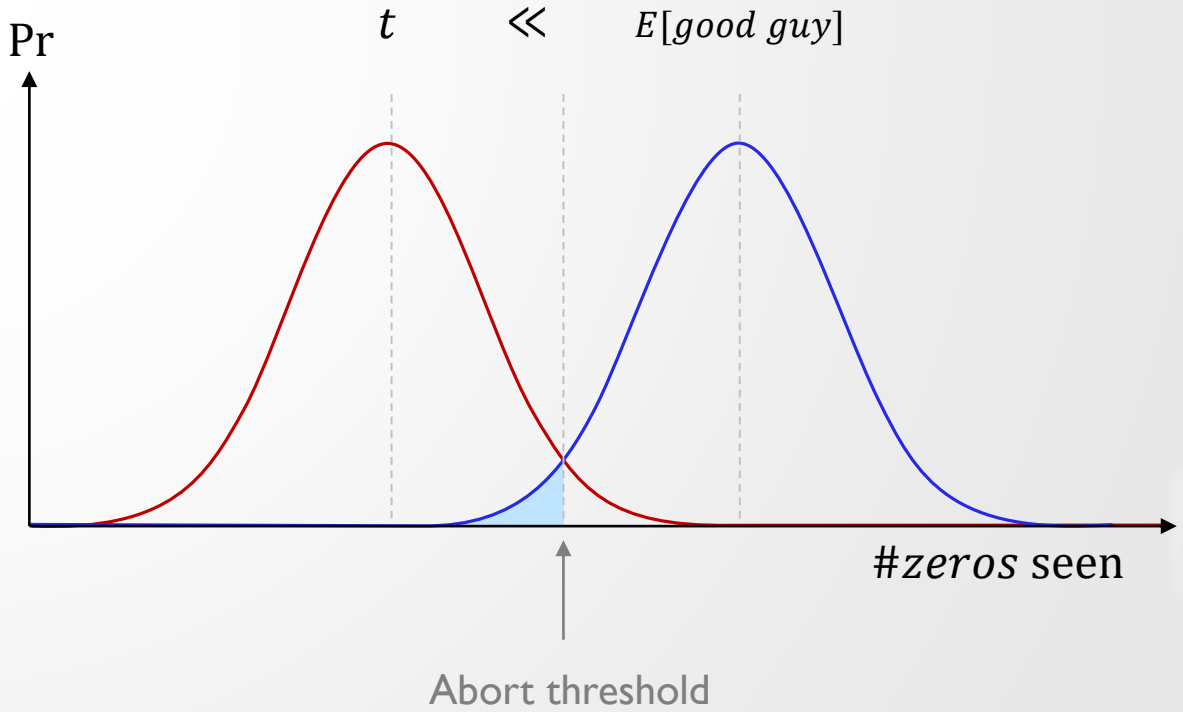
# Cut and Choose Parameters

[RindalRosulek17]

r Pr $Bad\ guy\ not\ caught \leq neg(\kappa)$ $Bad\ guy\ not\ caught$ $BBa$
$add\ gguuyy\ nnoott\ ccaauugghtt\ Bad\ guy\ not\ caught \leq nne$
$egg(\kappa\kappa)$ $Pr\ \ Bad\ guy\ not\ caught \leq neg(\kappa)$

*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:
  - $\Pr[good\ guy\ accused] \leq neg(\kappa)$
  - $Pr\ \ Bad\ guy\ not\ caught \leq neg(\kappa)$
- Use Chernoff Bounds
- Sufficient to check 1% of the OTs!



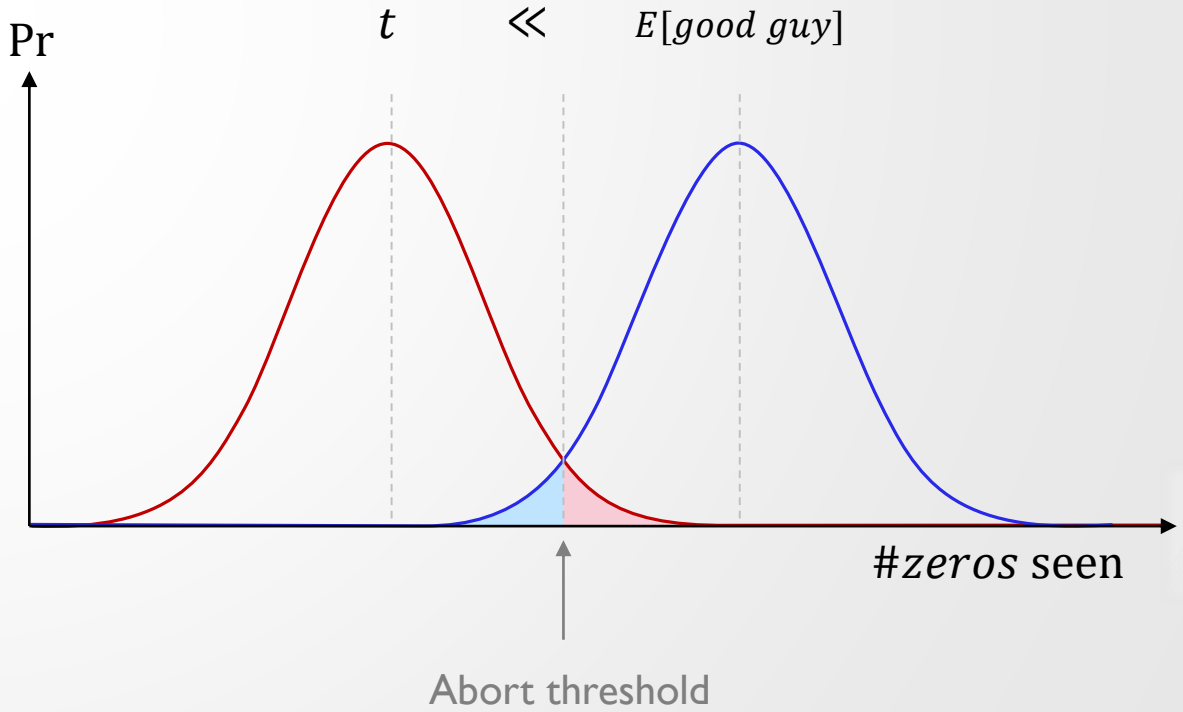
# Cut and Choose Parameters

[RindalRosulek17]

r Pr $Bad\ guy\ not\ caught \leq neg(\kappa)$ $Bad\ guy\ not\ caught$ $BBa$
$add\ gguuyy\ nnoott\ ccaauugghtt\ Bad\ guy\ not\ caught \leq nne$
$egg(\kappa\kappa)$ $Pr\ \ Bad\ guy\ not\ caught \leq neg(\kappa)$

*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:
  - $\Pr[good\ guy\ accused] \leq neg(\kappa)$
  - $Pr\ \ Bad\ guy\ not\ caught \leq neg(\kappa)$
- Use Chernoff Bounds
- Sufficient to check 1% of the OTs!



# Cut and Choose Parameters

[RindalRosulek17]

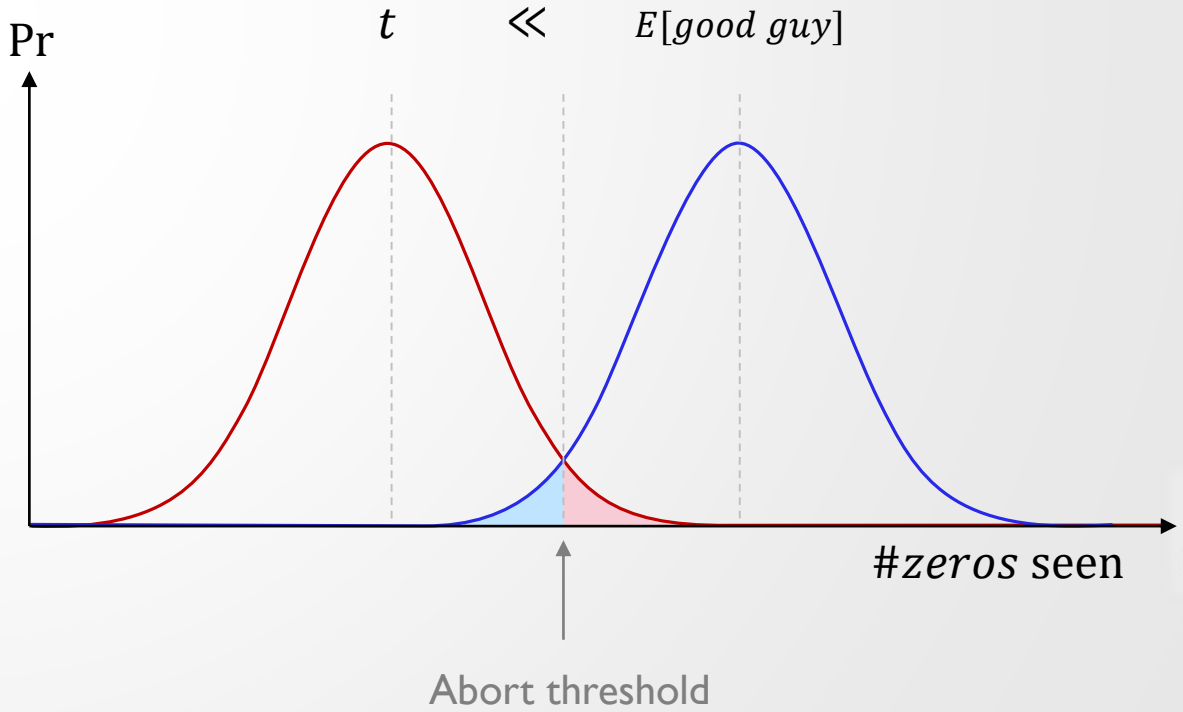
% of the OTs!

r Pr $Bad\ guy\ not\ caught \leq neg(\kappa)$ $Bad\ guy\ not\ caught$ $BBaadd\ ggu$ $uyy\ nnoott\ ccaauugghtt\ Bad\ guy\ not\ caught \leq nneegg(\kappa\kappa)$ $Pr\ \ Bad\ guy\ not\ caught \leq neg(\kappa)$

*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:
  - $\Pr[good\ guy\ accused] \leq neg(\kappa)$
  - $Pr\ \ Bad\ guy\ not\ caught \leq neg(\kappa)$

- Sufficient to check 1% of the OTs!
- Sufficient to check 1% of the OTs!



# Cut and Choose Parameters

[RindalRosulek17]

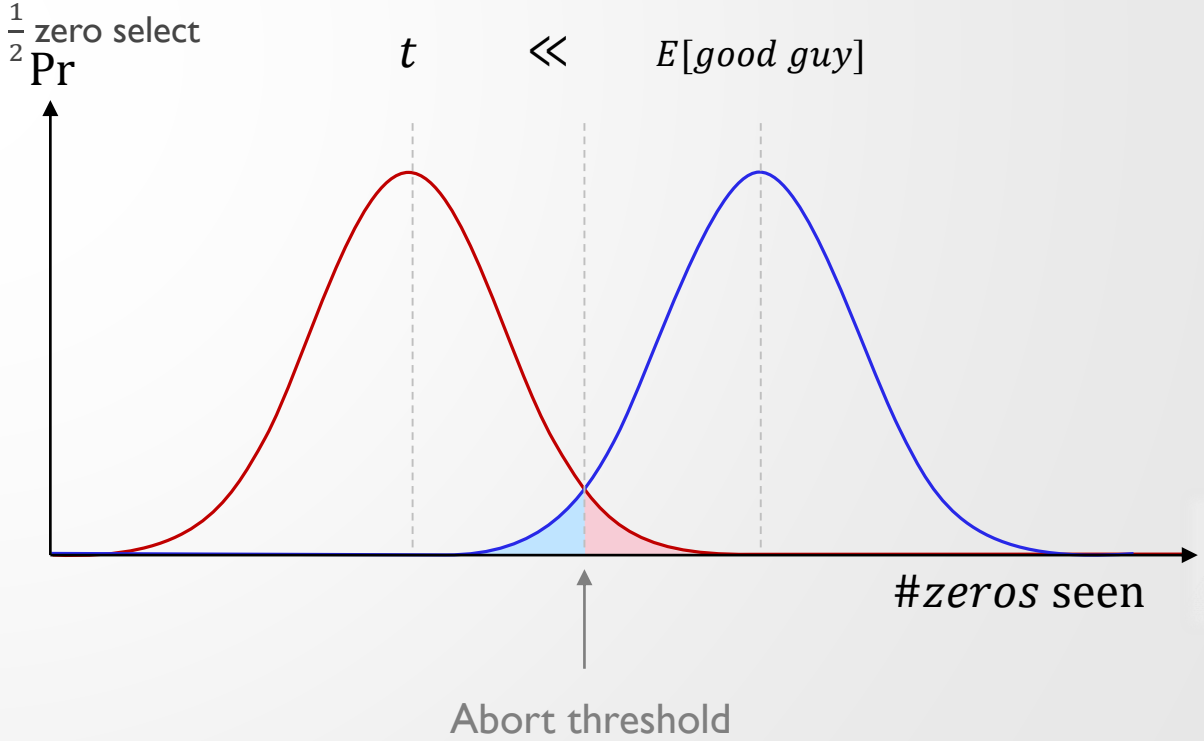
*% of the OTs*!

% of the OTs!

r Pr $Bad\ guy\ not\ caught \leq neg(\kappa)$ $Bad\ guy\ not\ caught$ $BBaadd\ gguuyy\ nnoot$ $t\ ccaauugghtt\ Bad\ guy\ not\ caught \leq nneegg(\kappa\kappa)$ Pr $Bad\ guy\ not\ caught \leq neg(\kappa)$

*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:
  - $\Pr[good\ guy\ accused] \leq neg(\kappa)$
  - $\Pr\ Bad\ guy\ not\ caught \leq neg(\kappa)$
- Sufficient to check 1% of the OTs!
- Sufficient to check 1% of the OTs!
- Sufficient to check 1% of the OTs!

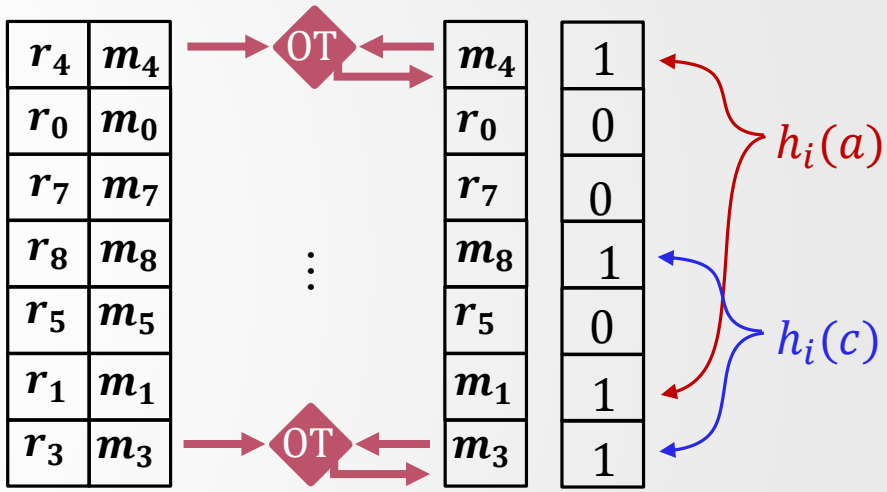


# Extracting $Y$ with Random Oracle

[RindalRosulek17]

Simulator must extract the effective input $Y$

$Y = \{a, c\}$




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

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

Output:

$$\hat{X} = \begin{Bmatrix} m_4 \oplus m_1, \\ m_2 \oplus m_3 \end{Bmatrix} \longrightarrow \hat{X} \cap \begin{Bmatrix} m_4 \oplus m_1, \\ m_8 \oplus m_3 \end{Bmatrix}$$

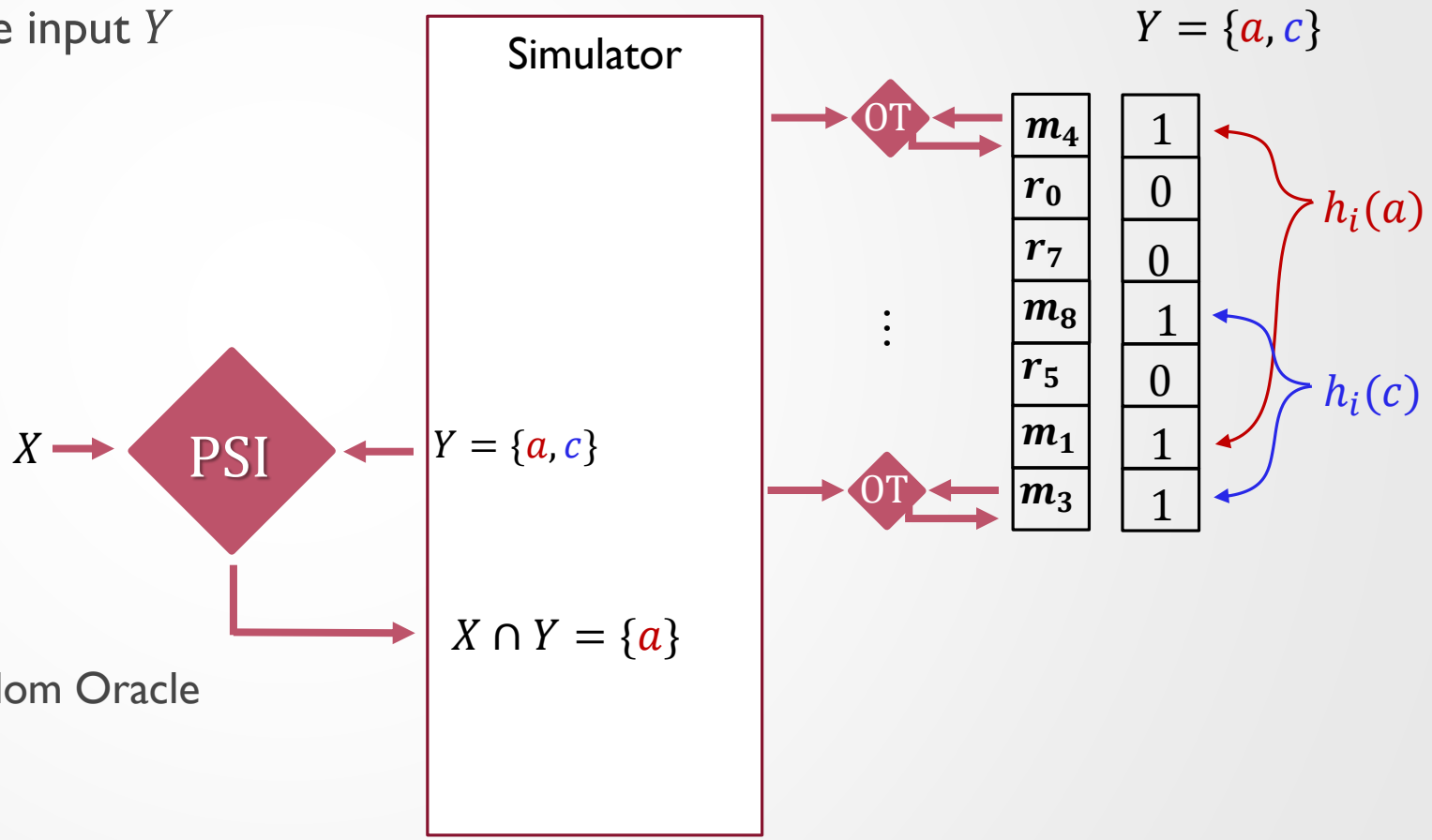
# Extracting $Y$ with Random Oracle

[RindalRosulek17]

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



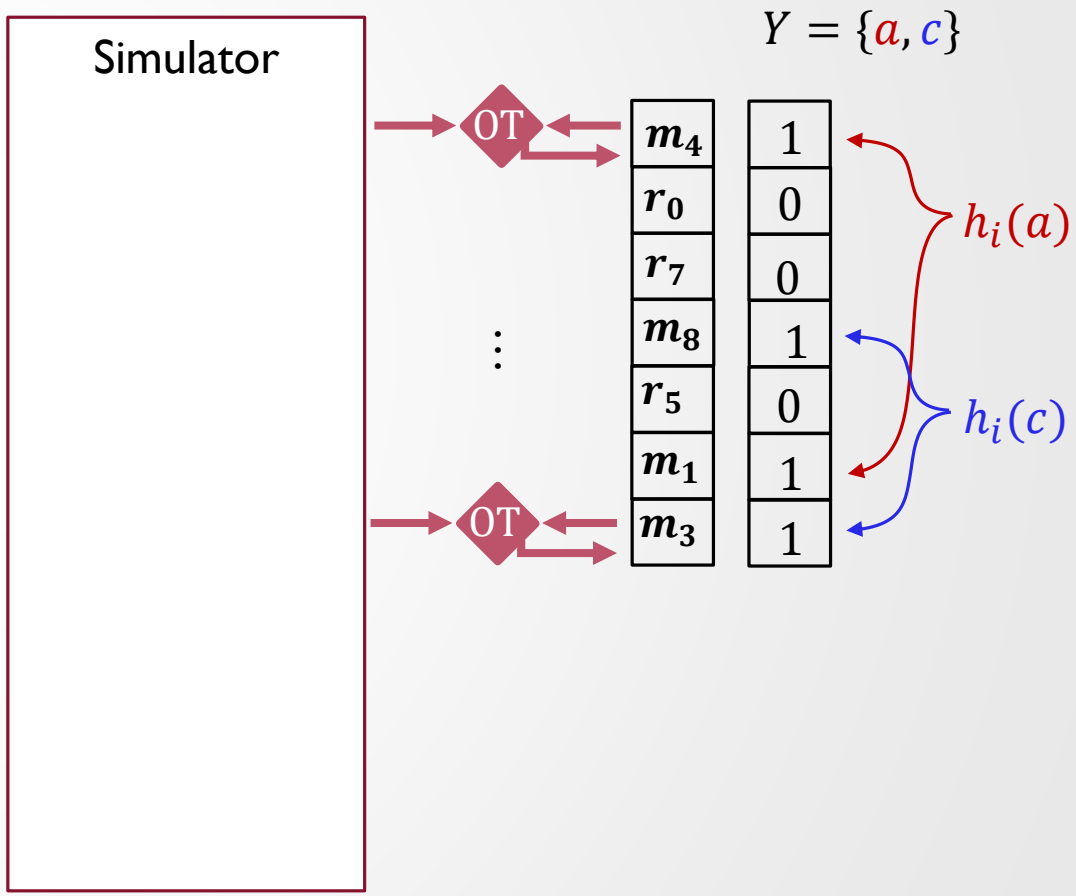
# Extracting $Y$ with Random Oracle

[RindalRosulek17]

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

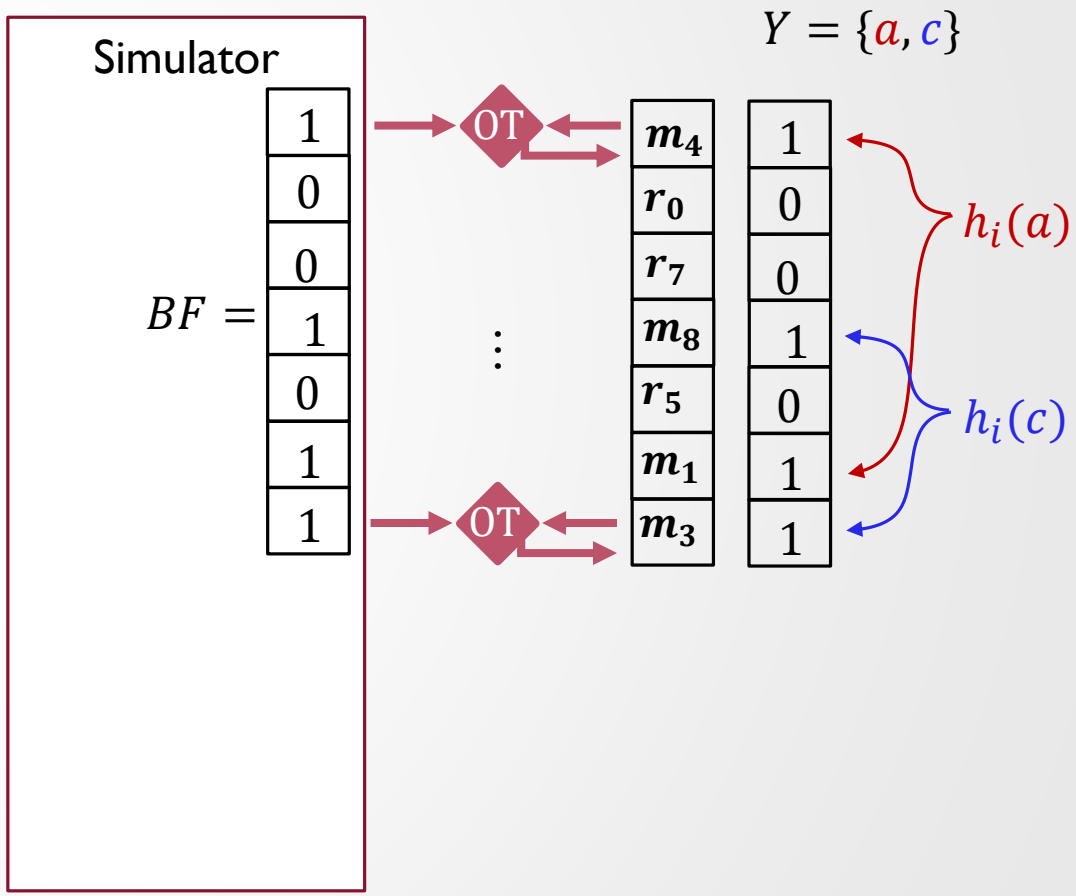
[RindalRosulek17]

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



# Extracting $Y$ with Random Oracle

[RindalRosulek17]

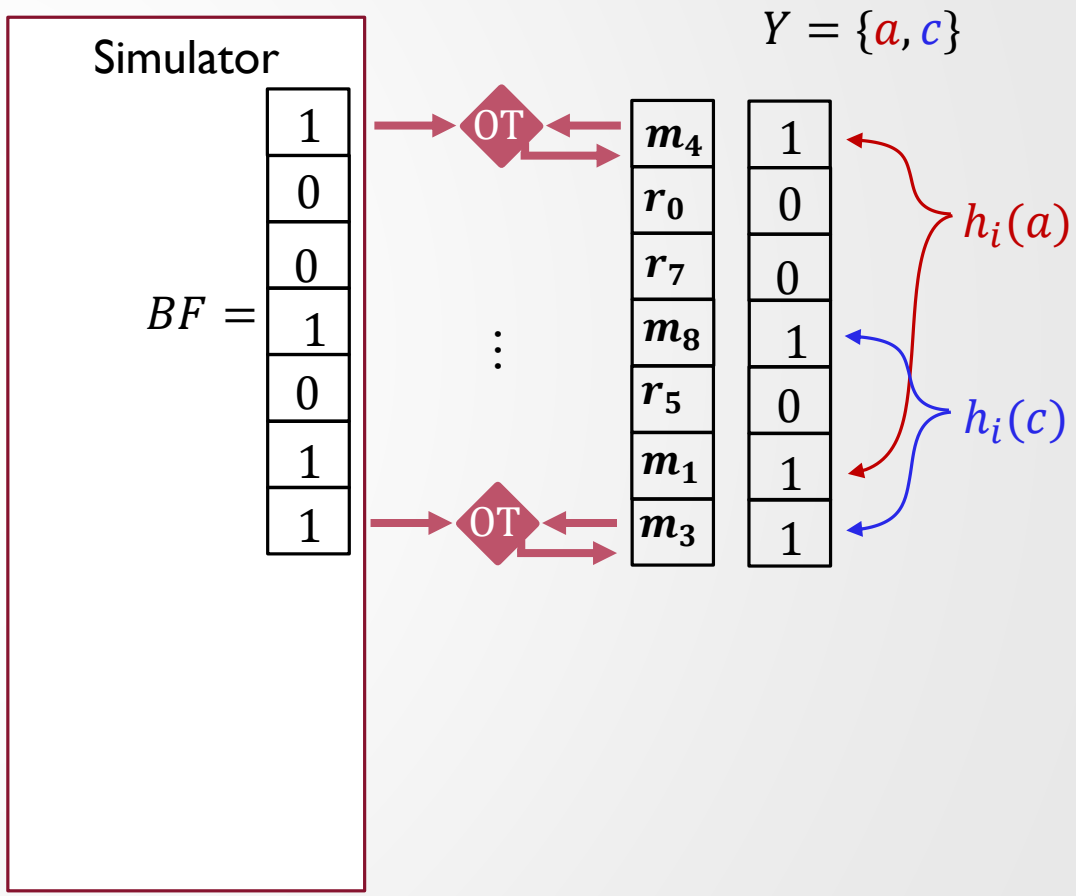
is not naturally invertible

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



# Extracting $Y$ with Random Oracle

[RindalRosulek17]

may be malformed…

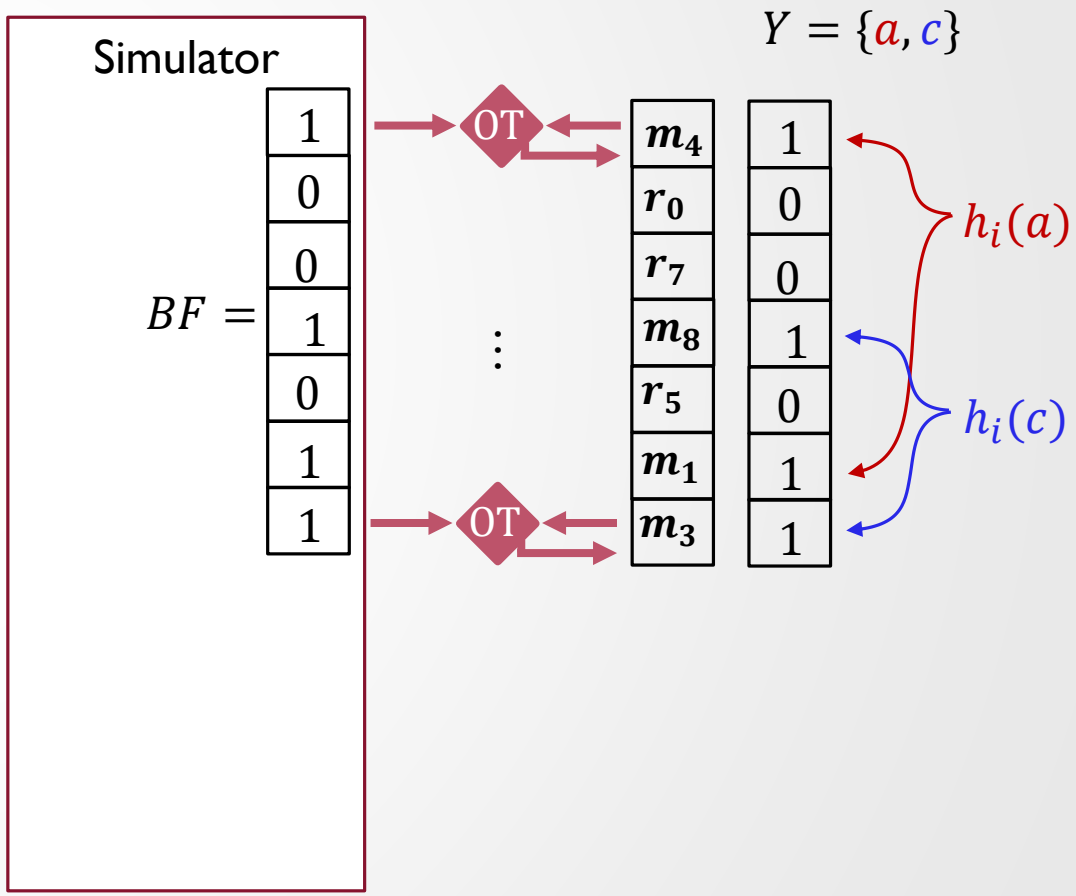
is not naturally invertible

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



# Extracting $Y$ with Random Oracle

[RindalRosulek17]

$h\ i\ ii\ h\ i\ (\cdot)$ as Random Oracle

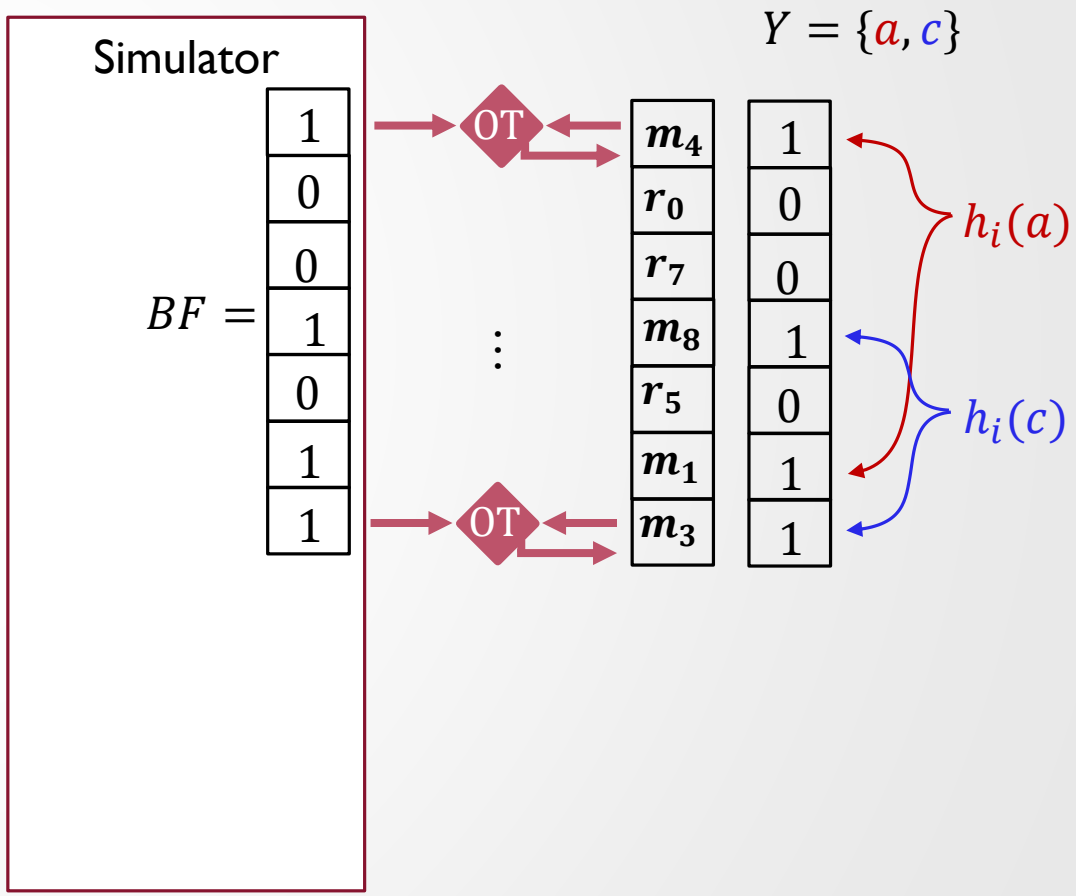
may be malformed…

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



# Extracting $Y$ with Random Oracle

[RindalRosulek17]

$h\ i\ ii\ h\ i\ (\cdot)$ as Random Oracle

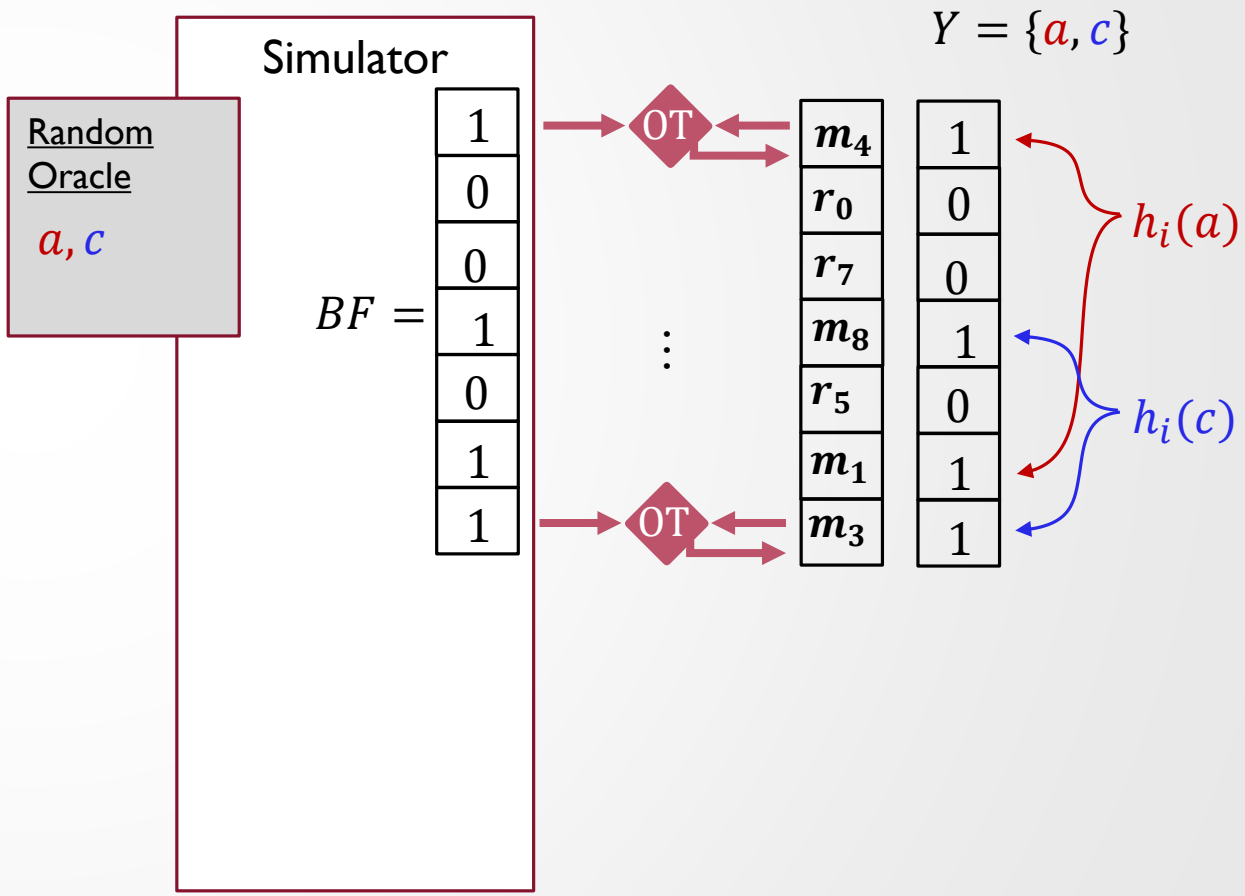
may be malformed…

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



# Extracting $Y$ with Random Oracle

[RindalRosulek17]

$h\ i\ ii\ h\ i\ (\cdot)$ as Random Oracle

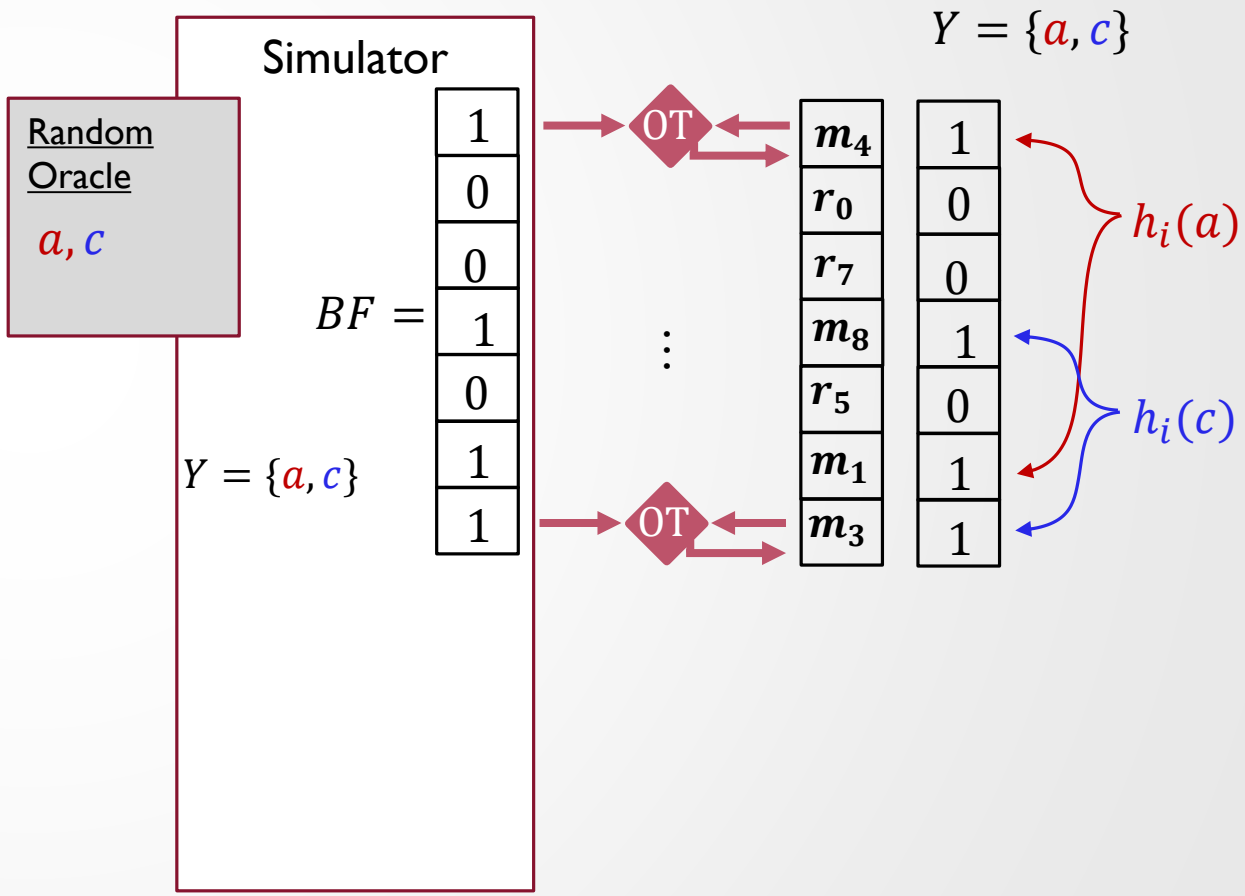
may be malformed…

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



# Extracting $Y$ with Random Oracle

[RindalRosulek17]

$h\ i\ ii\ h\ i\ (\cdot)$ as Random Oracle

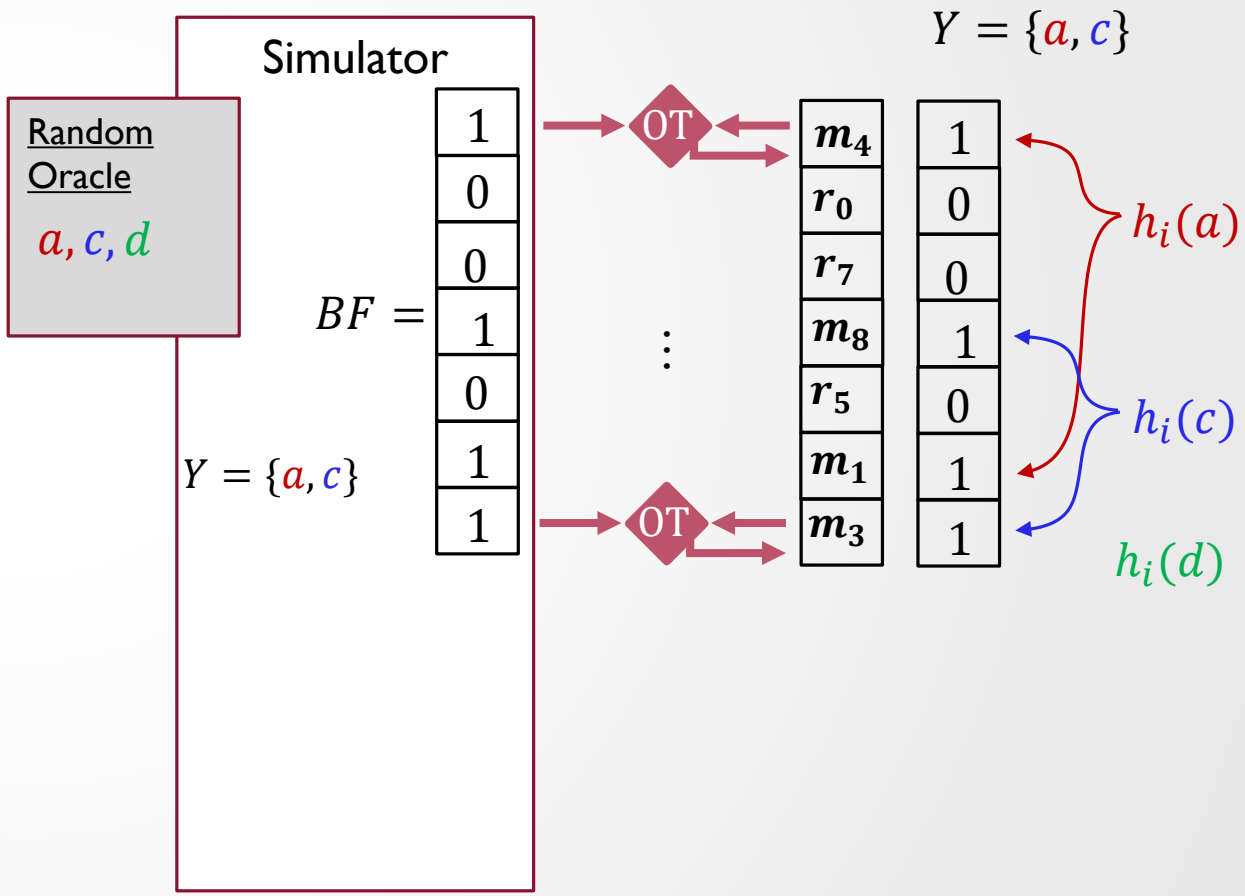
may be malformed…

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



# Extracting $Y$ with Random Oracle

[RindalRosulek17]

$h\ i\ ii\ h\ i\ (\cdot)$ as Random Oracle

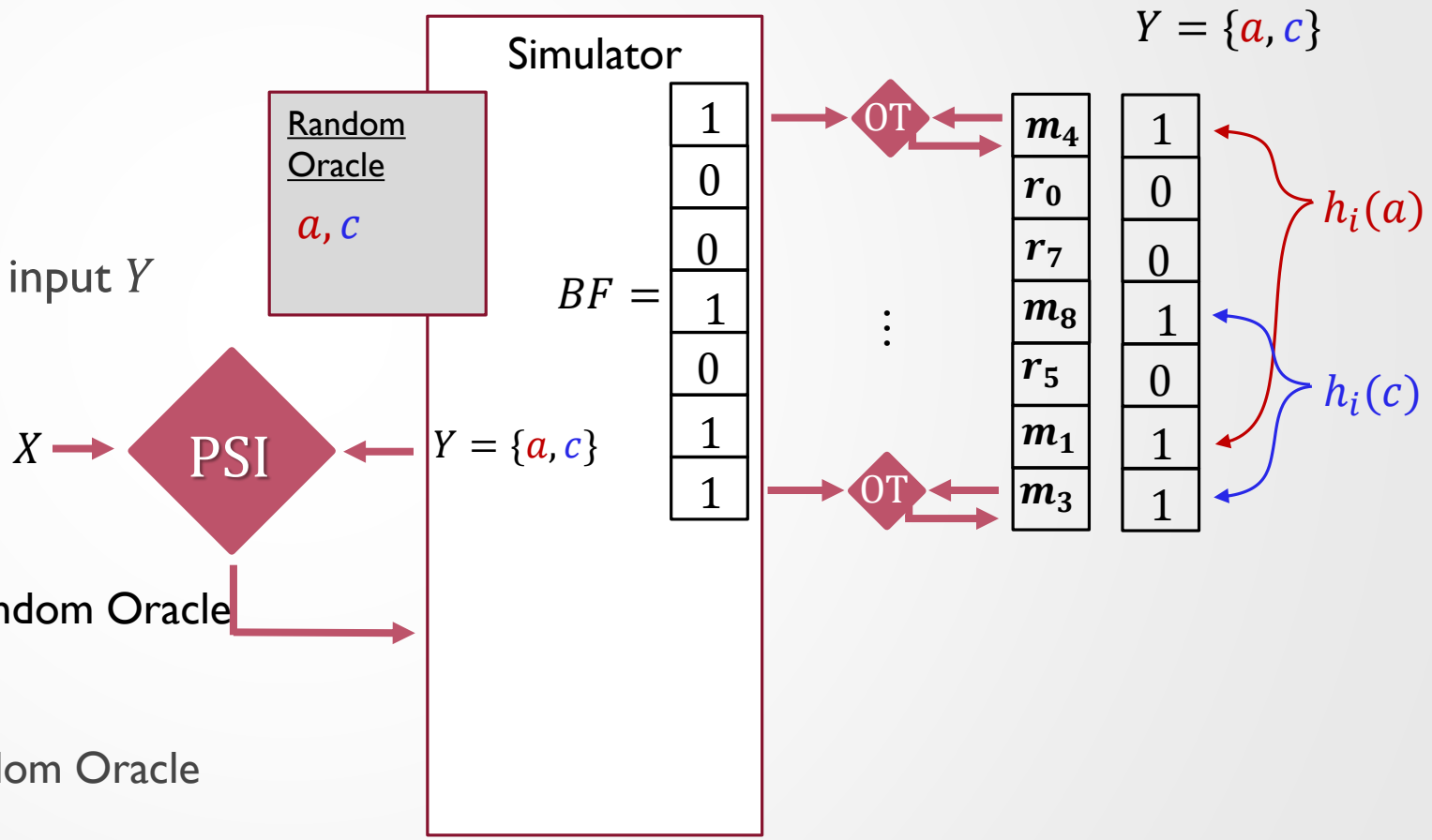
may be malformed…

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



# Extracting $Y$ with Random Oracle

[RindalRosulek17]

$h\ i\ ii\ h\ i\ (\cdot)$ as Random Oracle

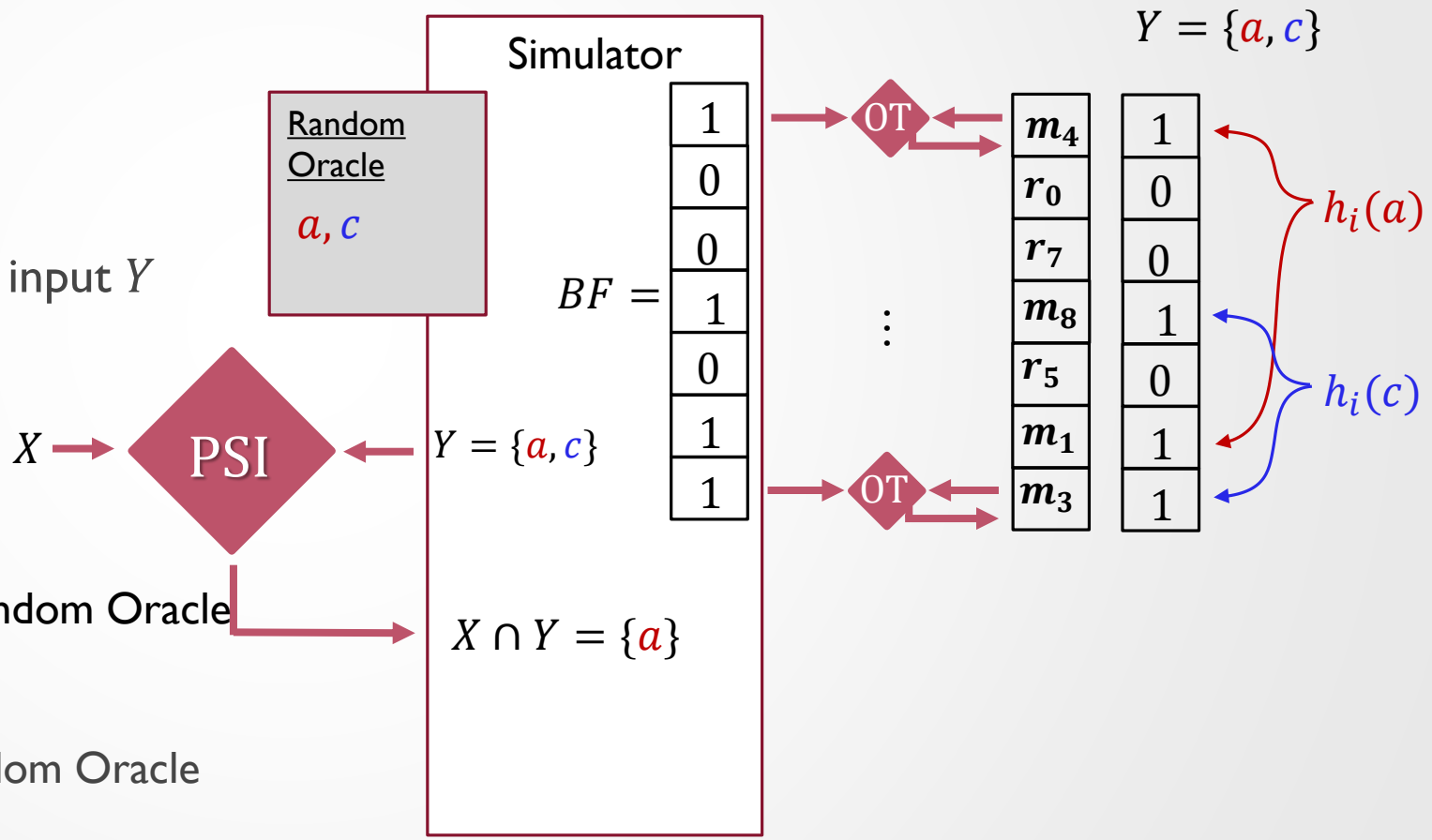
may be malformed…

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



# Extracting $Y$ with Random Oracle

[RindalRosulek17]

$h\ i\ ii\ h\ i\ (\cdot)$ as Random Oracle

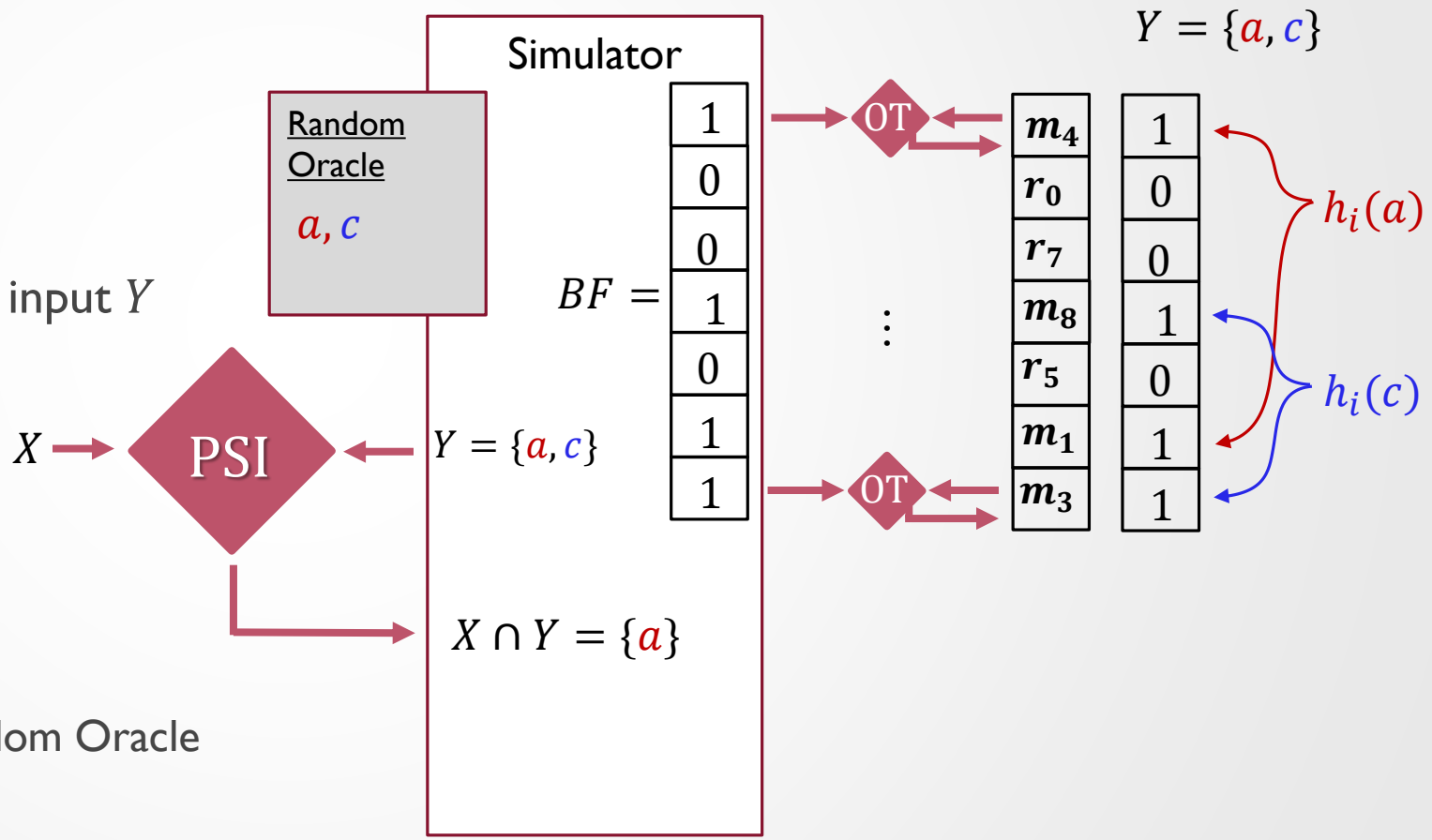
may be malformed…

is not naturally invertible

Simulator must extract the effective input $Y$

- Can extract OT select bits

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



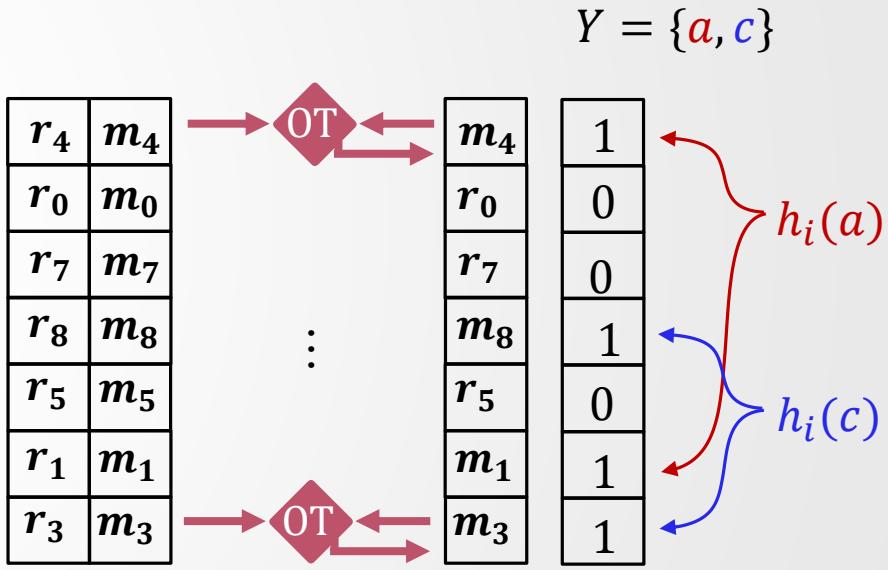
# Generalized Encodings

[RindalRosulek17]

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

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

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




$$\left\{\begin{matrix} \boldsymbol{m_4} \oplus \boldsymbol{m_1}, \\ \boldsymbol{m_8} \oplus \boldsymbol{m_3} \end{matrix}\right\}$$

# Generalized Encodings

[RindalRosulek17]

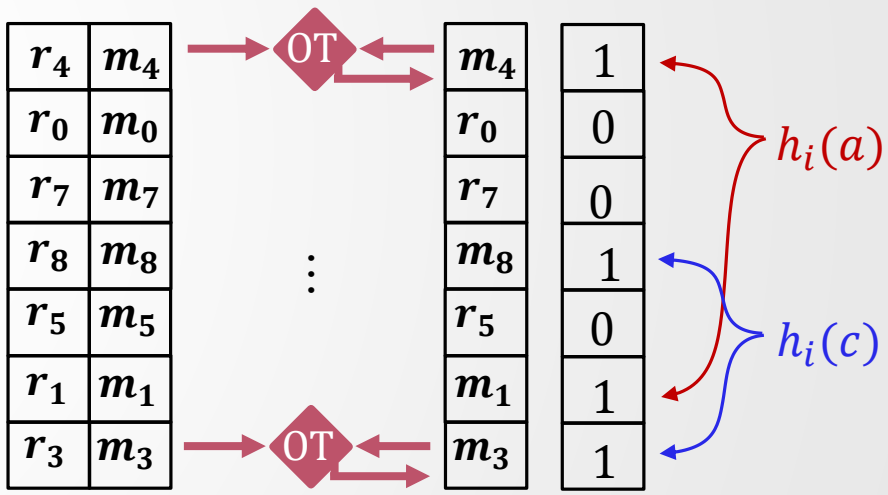
$F\ FF(\cdot)$

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

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

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

$Y = \{a, c\}$




$$\left\{\begin{matrix} \boldsymbol{m_4} \oplus \boldsymbol{m_1}, \\ \boldsymbol{m_8} \oplus \boldsymbol{m_3} \end{matrix}\right\}$$

# Generalized Encodings

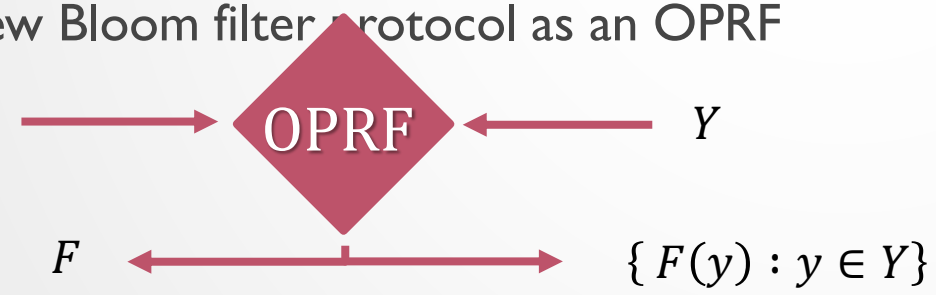
[RindalRosulek17]

$F$ $FF(\cdot)$

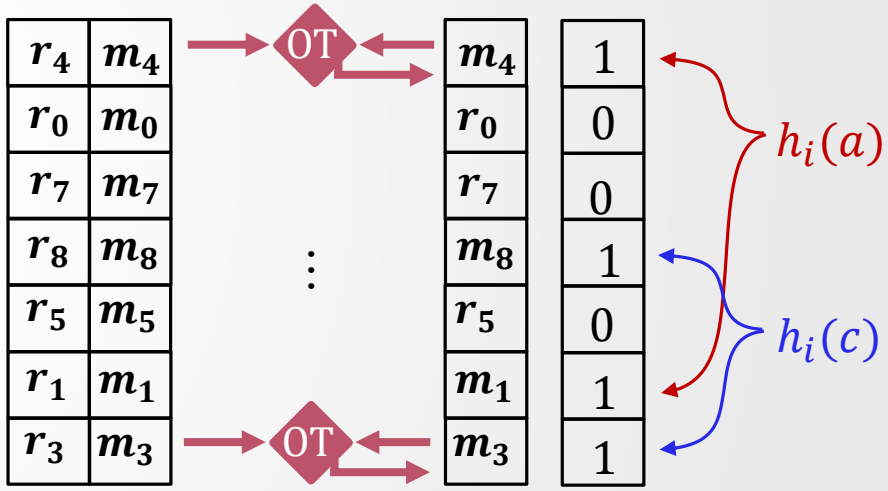
Bloom filter of size $\sim 2n\kappa$ allows a Receiver to insert $n$ items

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

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




$Y = \{a, c\}$




$$\left\{\begin{matrix} \boldsymbol{m_4} \oplus \boldsymbol{m_1}, \\ \boldsymbol{m_8} \oplus \boldsymbol{m_3} \end{matrix}\right\}$$

# Generalized Encodings

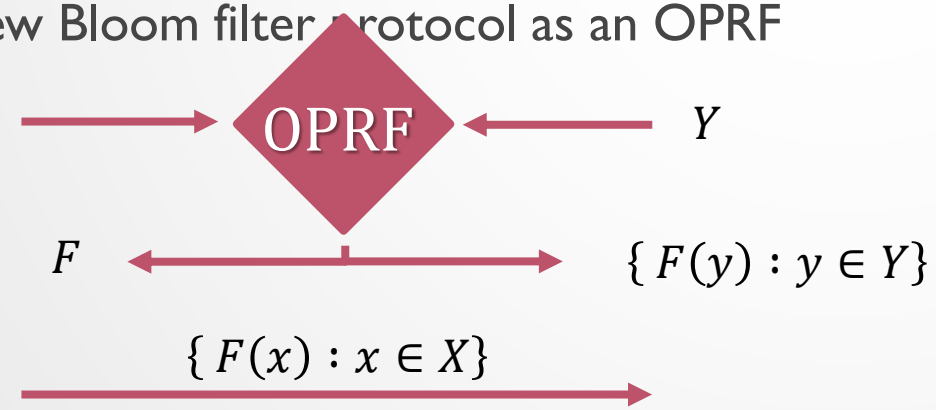
[RindalRosulek17]

$F$ $FF(\cdot)$

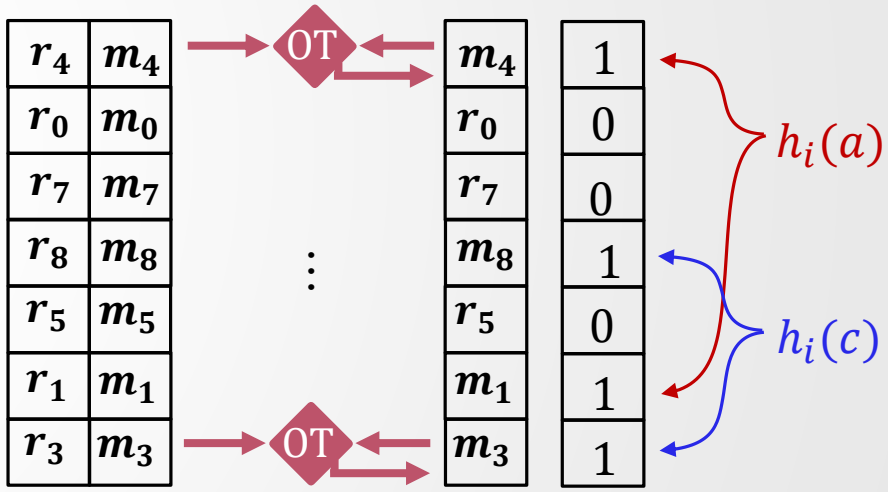
Bloom filter of size $\sim 2n\kappa$ allows a Receiver to insert $n$ items

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

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



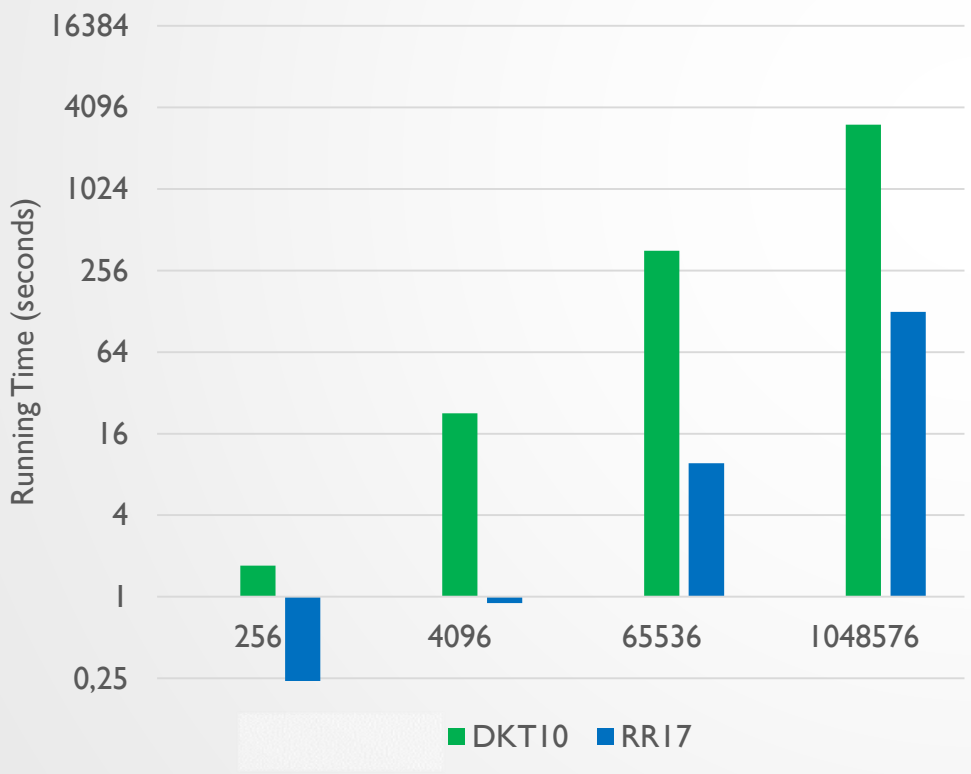

$Y = \{a, c\}$




$$\left\{\begin{matrix} \boldsymbol{m_4} \oplus \boldsymbol{m_1}, \\ \boldsymbol{m_8} \oplus \boldsymbol{m_3} \end{matrix}\right\}$$

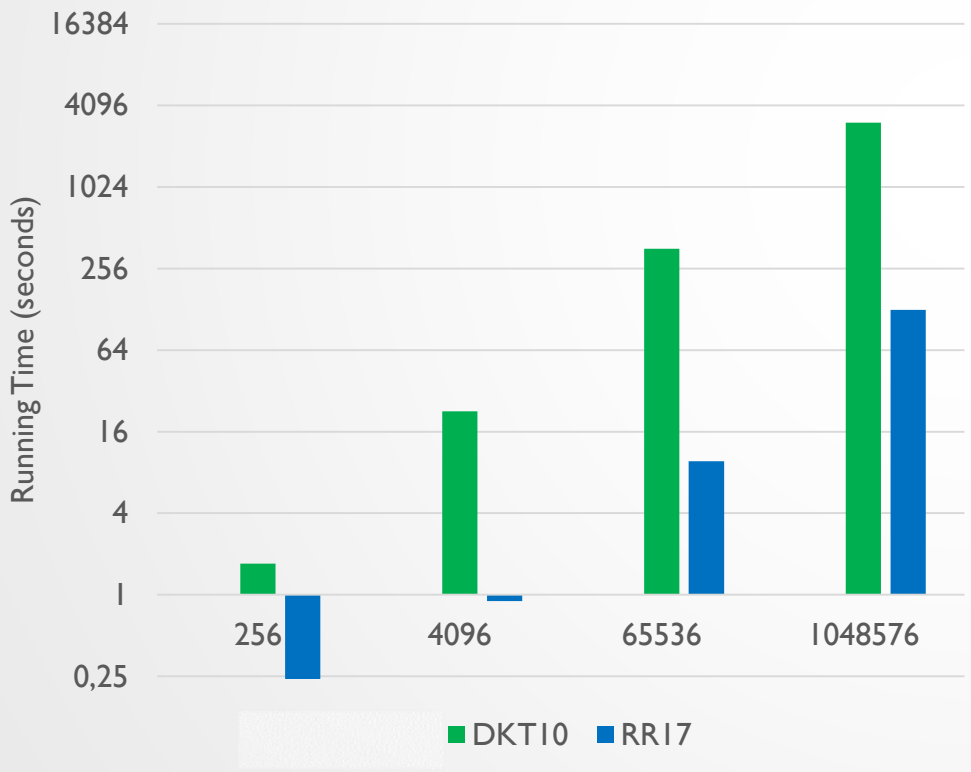
# Comparison – De Cristofaro, Kim, Tsudik10

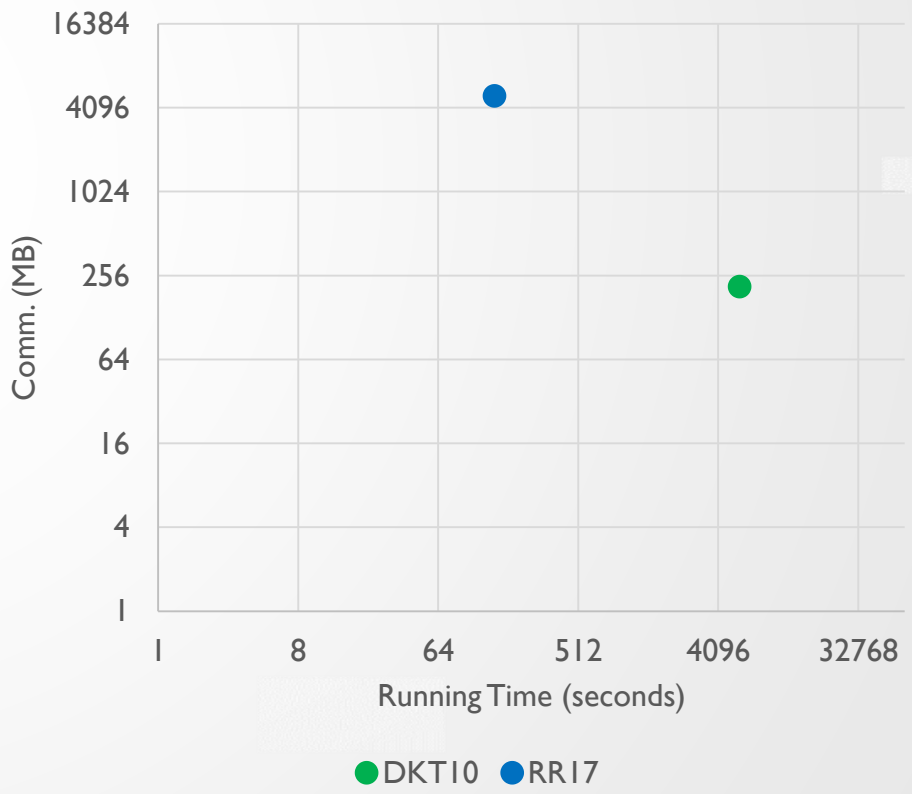
- DKT10 - Malicious Diffie-Hellman style approach: $x^{\alpha\beta} = y^{\beta\alpha}$



# Comparison – De Cristofaro, Kim, Tsudik10

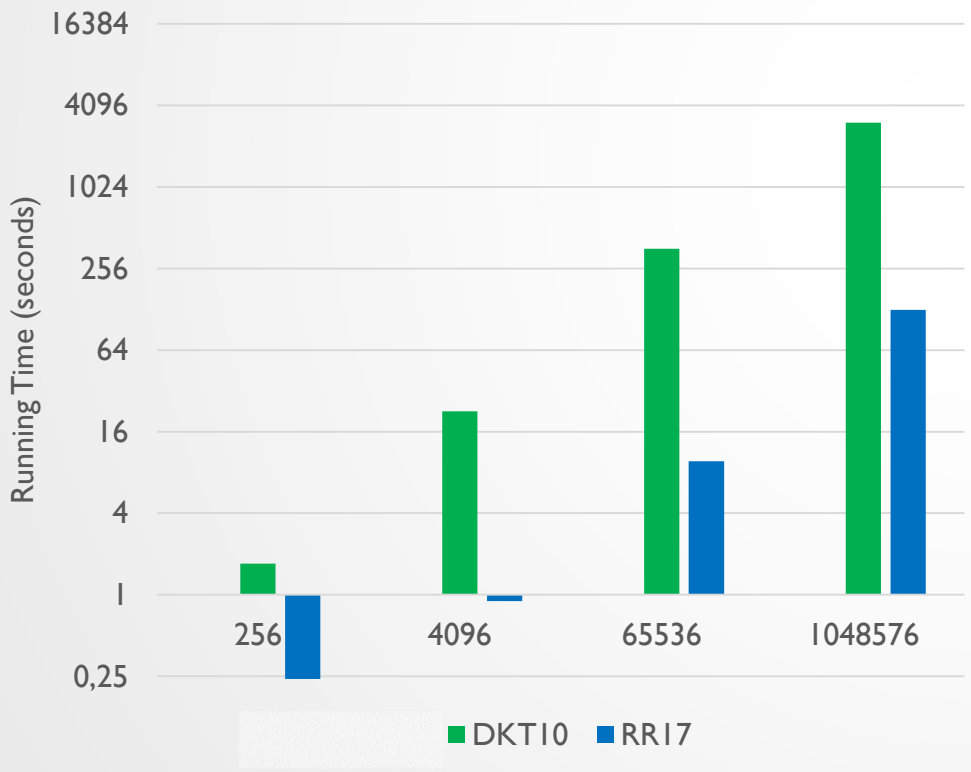
- DKT10 - Malicious Diffie-Hellman style approach: $x^{\alpha\beta} = y^{\beta\alpha}$

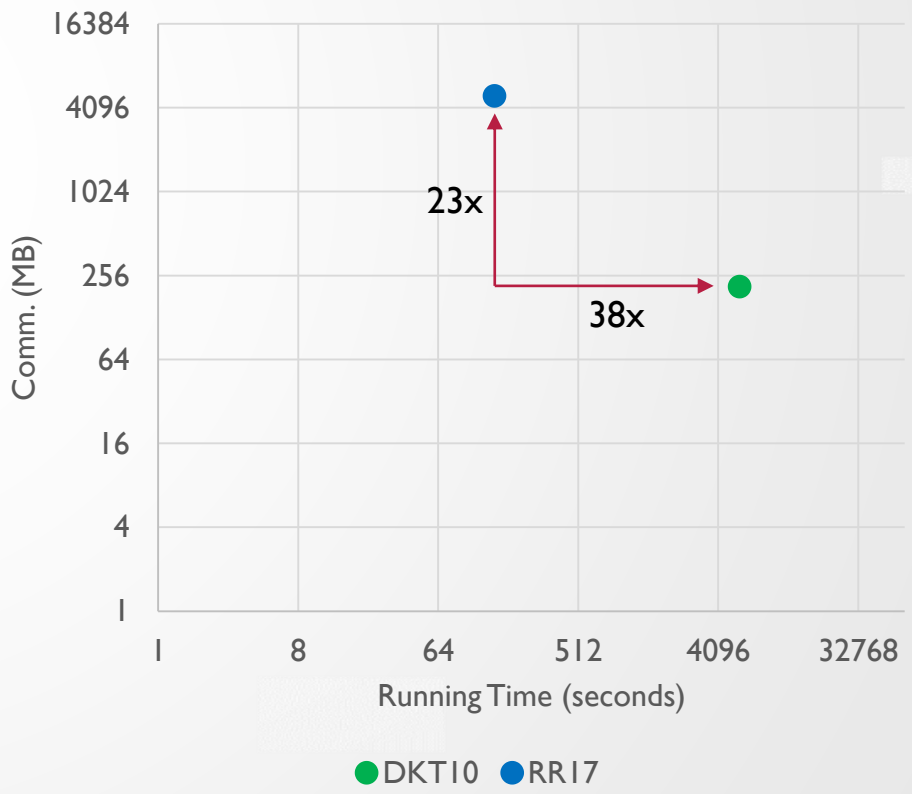


# Comparison – De Cristofaro, Kim, Tsudik10

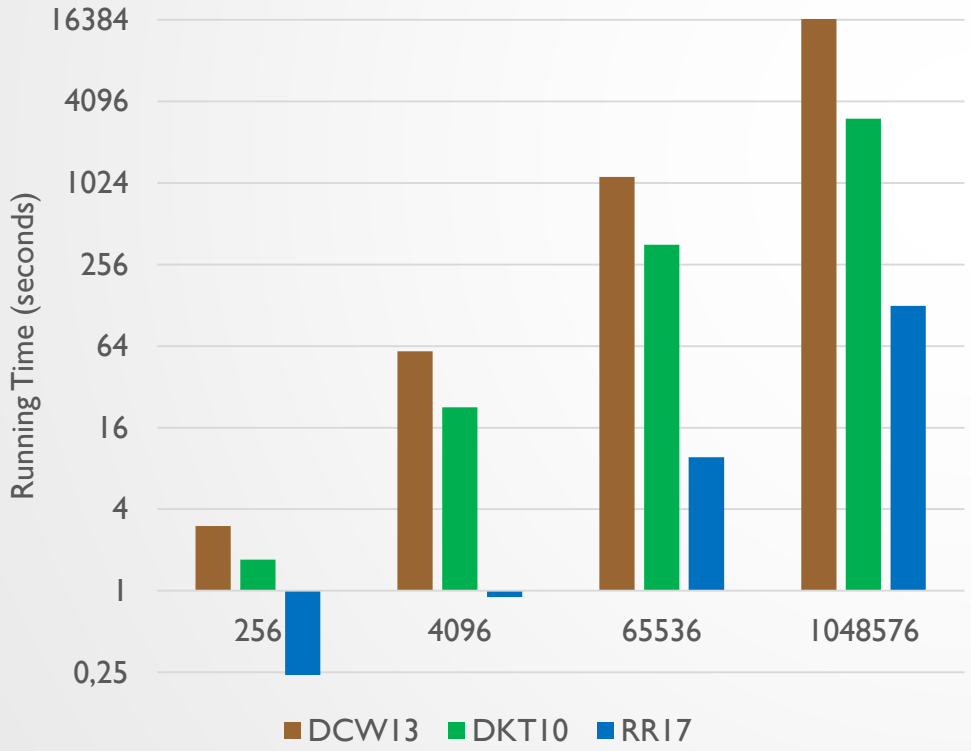
- DKT10 - Malicious Diffie-Hellman style approach: $x^{\alpha\beta} = y^{\beta\alpha}$

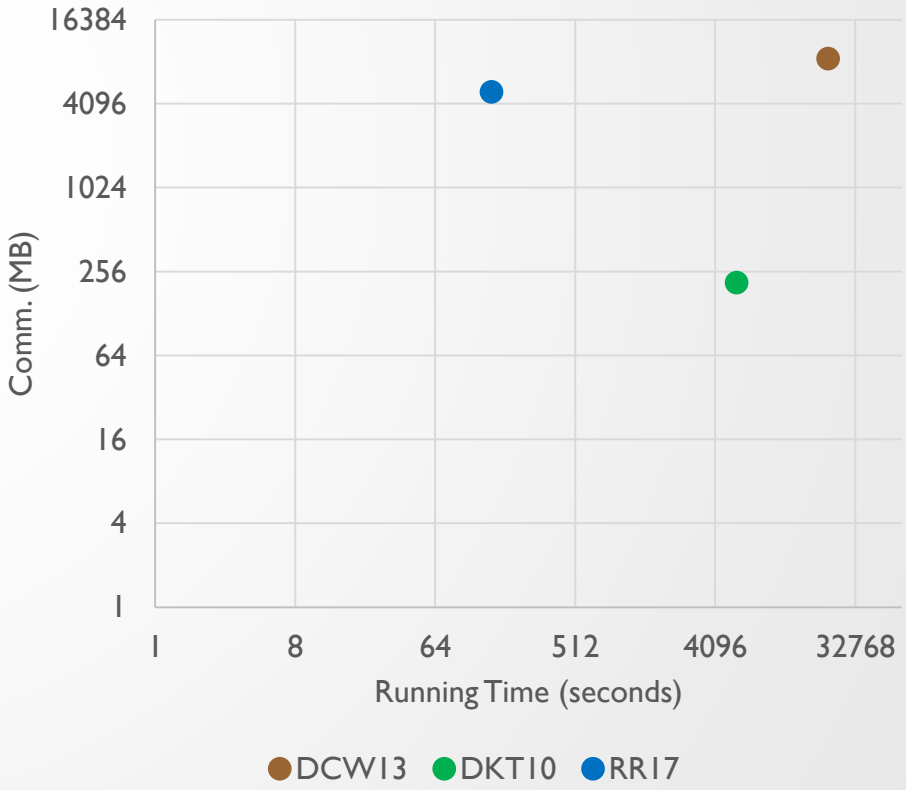


# Comparison – De Cristofaro, Kim, Tsudik10

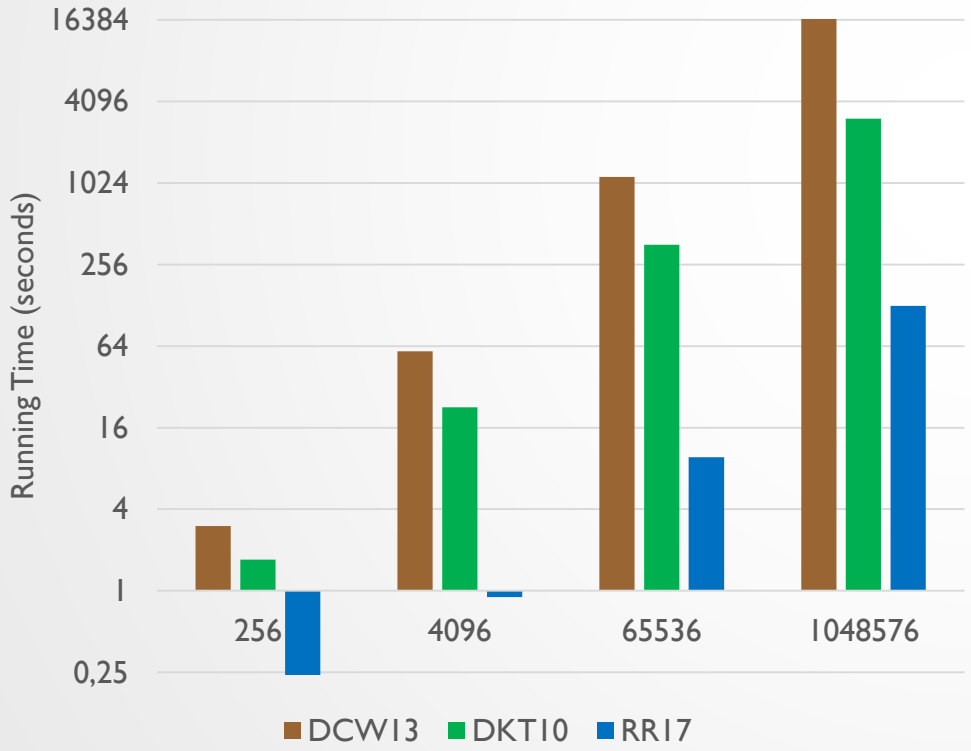
- DKT10 - Malicious Diffie-Hellman style approach: $x^{\alpha\beta} = y^{\beta\alpha}$

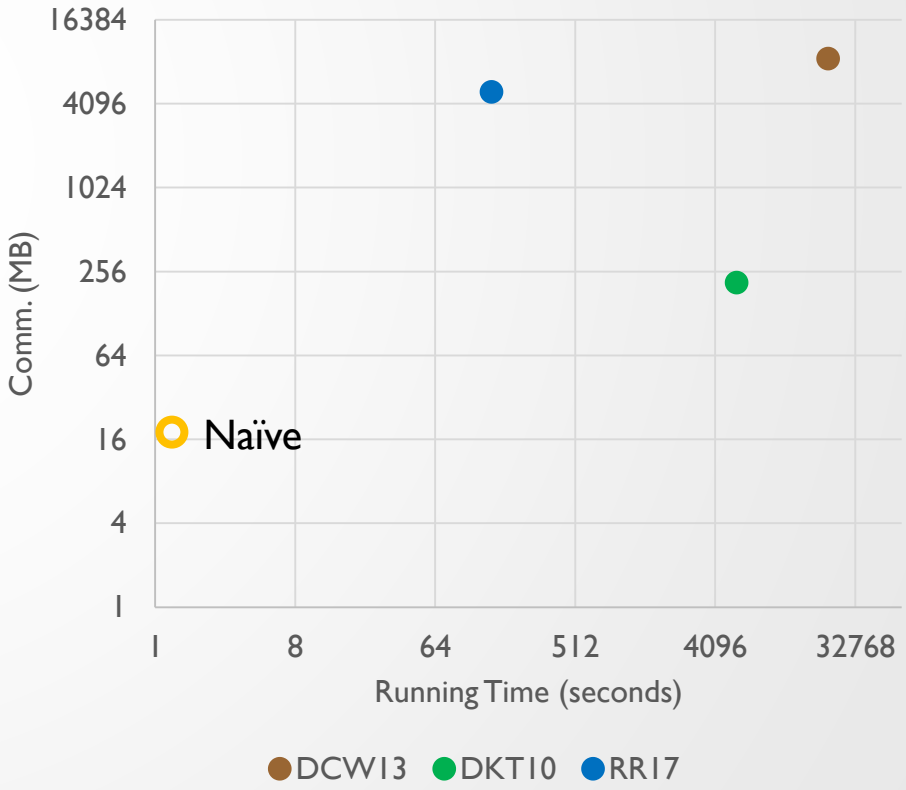


# Comparison – De Cristofaro, Kim, Tsudik10

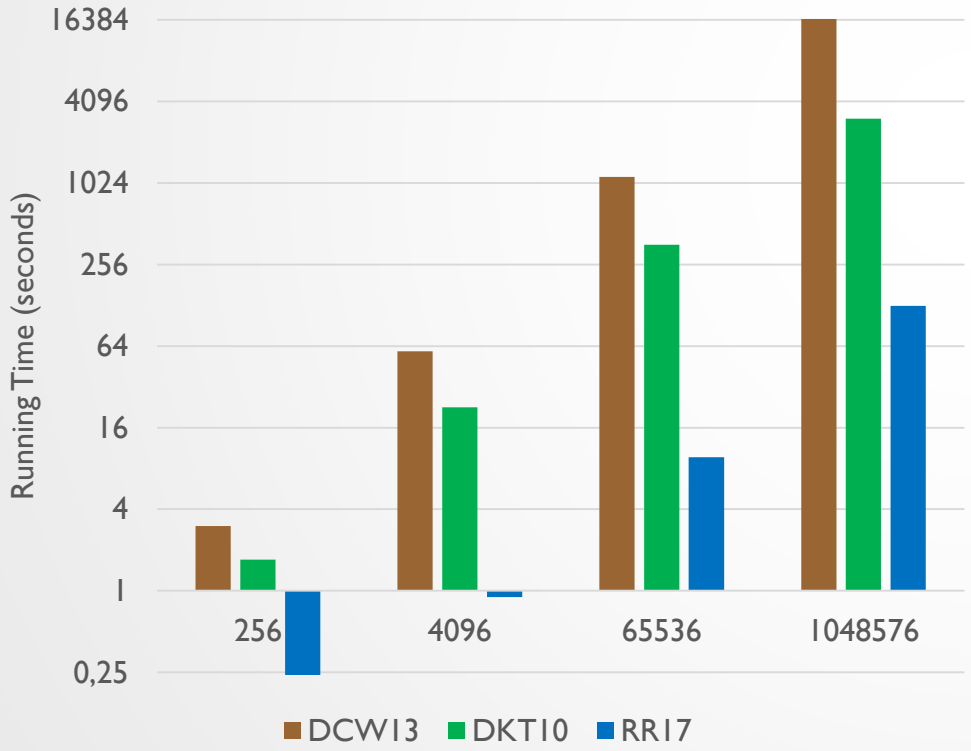
- DKT10 - Malicious Diffie-Hellman style approach: $x^{\alpha\beta} = y^{\beta\alpha}$

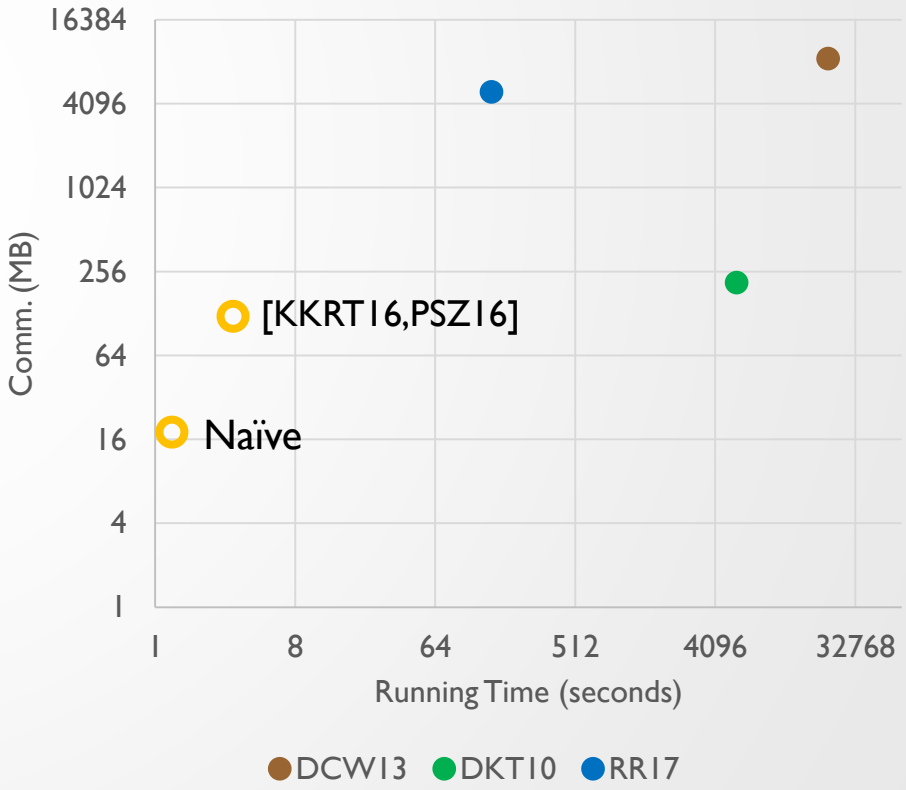


# Comparison – De Cristofaro, Kim, Tsudik10

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



# *The End*

**Peter Rindal**
Mike Rosulek

Oregon State UNIVERSITY OSU