Oblivious Transfer

Mike Rosulek

crypt@b-it 2018

\[ m_0, m_1 \rightarrow^\text{OT} c \rightarrow^\text{OT} m_c \]
OT recap

OT is . . .

- Necessary for MPC [Kilian]
- **Inherently expensive**: impossible using only cheap crypto (random oracle) [ImpagliazzoRudich89]
OT recap

OT is . . .

- Necessary for MPC [Kilian]
- **Inherently expensive**: impossible using only cheap crypto (random oracle) [ImpagliazzoRudich89]

**Today’s agenda**: reducing the cost of OT

1. **Precomputation**: can compute OTs even before you know your input!

2. **OT extension**: 128 OTs suffice for everything.
Random OT

Standard OT:
\[ m_0, m_1 \rightarrow OT \rightarrow c \rightarrow m_c \]

Random OT:
\[ m_0, m_1 \leftarrow OT \rightarrow c, m_c \]

Deterministic functionality; parties choose all inputs

Randomized functionality chooses \( m_0, m_1 \) uniformly.

Beaver Derandomization Theorem

There is a cheap protocol that securely evaluates an instance of standard OT using an instance of random OT.

Offline/online approach to 2PC:

- In offline preprocessing phase, generate many random OTs
- During online phase, OT inputs are determined — cheaply derandomize the offline OTs with Beaver’s trick.
Random OT

**Standard OT:**

\[ m_0, m_1 \rightarrow OT \leftarrow c \rightarrow m_c \]

Deterministic functionality; parties choose all inputs

**Random OT:**

\[ m_0, m_1 \leftrightarrow OT \rightarrow c, m_c \]

Randomized functionality chooses \( m_0, m_1, c \) uniformly.

Beaver Derandomization Theorem [Beaver91]

There is a cheap protocol that securely evaluates an instance of standard OT using an instance of random OT.

Offline/online approach to 2PC:

- In offline preprocessing phase, generate many random OTs
- During online phase, OT inputs are determined — cheaply derandomize the offline OTs with Beaver's trick.
Random OT

**Standard OT:**

\[
m_0, m_1 \rightarrow \text{OT} \leftarrow c \rightarrow m_c
\]

Deterministic functionality; parties choose all inputs

**Random OT:**

\[
m_0, m_1 \leftarrow \text{OT} \rightarrow c, m_c
\]

Randomized functionality chooses \(m_0, m_1, c\) uniformly.

**Beaver Derandomization Theorem** [Beaver91]

There is a **cheap** protocol that securely evaluates an instance of **standard OT** using an instance of **random OT**.
Random OT

Standard OT:
\[ m_0, m_1 \rightarrow \boxed{OT} \leftarrow c \rightarrow m_c \]
Deterministic functionality; parties choose all inputs

Random OT:
\[ m_0, m_1 \leftrightarrow \boxed{OT} \rightarrow c, m_c \]
Randomized functionality chooses \( m_0, m_1, c \) uniformly.

Beaver Derandomization Theorem [Beaver91]
There is a cheap protocol that securely evaluates an instance of standard OT using an instance of random OT.

Offline/online approach to 2PC:
- In offline preprocessing phase, generate many random OTs
- During online phase, OT inputs are determined — cheaply derandomize the offline OTs with Beaver’s trick.
Beaver Derandomization [Beaver91]

\[ m_0^$, m_1^$ \leftarrow \text{OT} \rightarrow c^$, m_c^$ \]

- - - - - - - - - - - - - - - - -
Beaver Derandomization [Beaver91]

\[ m_0^\$, m_1^\$ \xleftarrow{\text{OT}} c^\$, m_c^\$ \]

\( m_0, m_1 \) offline online \( c \) compute \( m_c \) ?

Idea: Alice can use \( m_0^\$ \) and \( m_1^\$ \) as one-time pads to mask \( m_0, m_1 \); if \( c = c^\$ \) this works: Bob can decrypt only \( m_c \) (no info about \( m_1 \)); if \( c, c^\$ \), Bob learns wrong \( m_c \) unless Alice swaps \( m_0^\$ \), \( m_1^\$ \).

Solution: Bob says whether \( c = c^\$ \) (safe: Alice has no info about \( c \) ).
Beaver Derandomization [Beaver91]

**Idea:** Alice can use $m_0^\$ \text{ and } m_1^\$ \text{ as one-time pads to mask } m_0, m_1$
Beaver Derandomization [Beaver91]

\[ m_0^\$, m_1^\$ \leftarrow \text{OT} \rightarrow c^\$, m_c^\$ \]

\[ m_0, m_1 \quad \text{offline} \quad \text{online} \quad c \quad (= c^\$) \]

\[ x_0 = m_0^\$ \oplus m_0 \]
\[ x_1 = m_1^\$ \oplus m_1 \]

compute \( x_c \oplus m_c^\$ \)
\[ = x_c \oplus m_c^\$ = m_c \]

- **Idea:** Alice can use \( m_0^\$ \) and \( m_1^\$ \) as one-time pads to mask \( m_0, m_1 \)
- If \( c = c^\$ \) this works: Bob can decrypt only \( m_c \) (no info about \( m_{1-c} \))
**Beaver Derandomization** [Beaver91]

\[ m_0^\$, m_1^\$ \leftrightarrow \text{OT} \rightarrow c^\$, m_c^\$ \]

- offline
- online

\[ m_0, m_1 \quad c \ (\neq c^\$) \]

\[ x_0 = m_1^\$ \oplus m_0 \]
\[ x_1 = m_0^\$ \oplus m_1 \]

compute \[ x_c \oplus m_c^\$ \]
\[ = x_c \oplus m_1^\$ \oplus c = m_c \]

**Idea:** Alice can use \( m_0^\$ \) and \( m_1^\$ \) as one-time pads to mask \( m_0, m_1 \)

- If \( c = c^\$ \) this works: Bob can decrypt only \( m_c \) (no info about \( m_{1-c} \))
- If \( c \neq c^\$ \) Bob learns wrong \( m \) unless Alice swaps \( m_0^\$, m_1^\$.}
Beaver Derandomization [Beaver91]

- Idea: Alice can use $m_0^\$ and $m_1^\$ as one-time pads to mask $m_0, m_1$
- If $c = c^\$ this works: Bob can decrypt only $m_c$ (no info about $m_{1-c}$)
- If $c \neq c^\$ Bob learns wrong $m$ unless Alice swaps $m_0^\$, $m_1^\$.
- Solution: Bob says whether $c = c^\$ (safe: Alice has no info about $c^\$)
Beaver Derandomization [Beaver91]

Idea: Alice can use $m_0^s$ and $m_1^s$ as one-time pads to mask $m_0, m_1$

If $c = c^s$ this works: Bob can decrypt only $m_c$ (no info about $m_{1-c}$)

If $c \neq c^s$ Bob learns wrong $m$ unless Alice swaps $m_0^s, m_1^s$.

Solution: Bob says whether $c = c^s$ (safe: Alice has no info about $c^s$)

$$m_0, m_1 \leftarrow \text{OT} \rightarrow c^s, m_c^s$$

$$d = c \oplus c^s$$

$$x_0 = m_d^s \oplus m_0$$
$$x_1 = m_d^s \oplus m_1$$

compute $x_c \oplus m_c^s$

$$= x_c \oplus m_{c \oplus d}^s = m_c$$
Beaver Derandomization [Beaver91]

- **Offline cost:** same as before (1 OT instance)
- **Online cost:** simple XORs
E paucis plura

from a few, many
Oblivious Transfer is inherently expensive:

- Impossible using only cheap crypto (random oracle)

[ImpagliazzoRudich89]

Is there an analog of “hybrid encryption” for OT?

Public-key encryption is inherently expensive:

- Impossible using only cheap crypto (random oracle)

[PKE cost be minimized with hybrid encryption:
- Use (expensive) PKE to encrypt short
- Use (cheap) symmetric-key encryption with key $s$ to encrypt long $M$

$\lambda$ instances of OT + cheap SKE = $N$ instances of OT

$\lambda$ instances of OT + cheap SKE = PKE of $\lambda$ bits + cheap SKE = PKE of $N$ bits.
An analogy from encryption

Oblivious Transfer is inherently expensive:
  ▶ Impossible using only cheap crypto (random oracle)

Public-key encryption is inherently expensive:
  ▶ Impossible using only cheap crypto (random oracle)

[ImpagliazzoRudich89]
An analogy from encryption

**Oblivious Transfer** is inherently expensive:
- Impossible using only cheap crypto (random oracle)  
  [ImpagliazzoRudich89]

**Public-key encryption** is inherently expensive:
- Impossible using only cheap crypto (random oracle)  
  [ImpagliazzoRudich89]

PKE cost be **minimized** with hybrid encryption:
- Use (expensive) PKE to encrypt short $s$
- Use (cheap) symmetric-key encryption *with key* $s$ to encrypt long $M$

PKE of $\lambda$ bits + cheap SKE = PKE of $N$ bits
An analogy from encryption

**Oblivious Transfer is inherently expensive:**
- Impossible using only cheap crypto (random oracle) [ImpagliazzoRudich89]

*Is there an analog of “hybrid encryption” for OT?*

\[ \lambda \text{ instances of OT} + \text{cheap SKE} = N \text{ instances of OT} \]

**Public-key encryption is inherently expensive:**
- Impossible using only cheap crypto (random oracle) [ImpagliazzoRudich89]

PKE cost be **minimized** with hybrid encryption:
- Use (expensive) PKE to encrypt short \( s \)
- Use (cheap) symmetric-key encryption *with key* \( s \) to encrypt long \( M \)

\[ \text{PKE of } \lambda \text{ bits} + \text{cheap SKE} = \text{PKE of } N \text{ bits} \]
Beaver OT extension [Beaver96]

Key insight: Yao’s protocol requires only # of OTs proportional to function’s input length
Beaver OT extension [Beaver96]

**Key insight:** Yao’s protocol requires only # of OTs proportional to function’s input length

**Beaver protocol:** Run the following 2PC using Yao:

- Use $s_A \oplus s_B$ as pseudorandom seed to:
  - Sample $2n$ random strings $(m_1,0, m_1,1), \ldots, (m_n,0, m_n,1)$.
  - Sample $n$-bit random string $r$.

- $s_B \in \{0, 1\}^\lambda$
**Beaver OT extension** [Beaver96]

**Key insight:** Yao’s protocol requires only # of OTs proportional to function’s **input length**

**Beaver protocol:** Run the following 2PC using Yao:

- Use $s_A \oplus s_B$ as pseudorandom seed to:
  - Sample $2n$ random strings $(m_1,0,m_1,1), \ldots, (m_n,0,m_n,1)$.
  - Sample $n$-bit random string $r$

- # OTs = input length = $\lambda$
- Output provides $n \gg \lambda$ instances of OT (random strings + choice bits)
- Impractical **feasibility** result (2PC evaluation of a PRG circuit)
IKNP protocol [IshaiKilianNissimPetrank03]

- Bob has input $r$

<table>
<thead>
<tr>
<th>$r$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>

Bob
Bob has input $r \Rightarrow$ extend to matrix
Bob has input $r \Rightarrow$ extend to matrix and secret share as $(T, T')$
IKNP protocol [IshaiKilianNissimPetrank03]

- Bob has input $r$ ⇒ extend to matrix and secret share as $(T, T')$
- Alice chooses random string $s$
**IKNP protocol**  [IshaiKilianNissimPetrank03]

<table>
<thead>
<tr>
<th>$s = 0$</th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</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>
</tr>
<tr>
<td>0</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>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Bob has input $r$ ⇒ extend to matrix and secret share as $(T, T')$

Alice chooses random string $s$

OT for each **column** ⇒ Alice obtains matrix $Q$
IKNP protocol [IshaiKilianNissimPetrank03]

Bob has input $r \Rightarrow$ extend to matrix and secret share as $(T, T')$

Alice chooses random string $s$

OT for each **column** $\Rightarrow$ Alice obtains matrix $Q$
IKNP protocol

[ShaiKilianNissimPetrank03]

Bob has input $r$ ⇒ extend to matrix and secret share as $(T, T')$

Alice chooses random string $s$

OT for each column ⇒ Alice obtains matrix $Q$
IKNP protocol [IshaiKilianNissimPetrank03]

Bob has input $r \Rightarrow$ extend to matrix and secret share as $(T, T')$

Alice chooses random string $s$

OT for each column $\Rightarrow$ Alice obtains matrix $Q$

Alice

$\begin{array}{cccccc}
  s = 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 \\
  1 & 0 & 0 & 1 \\
  1 & 0 & 1 & 1 \\
  0 & 1 & 1 & 0 \\
  1 & 1 & 0 & 1 \\
  1 & 0 & 0 & 1 \\
  1 & 0 & 1 & 0 \\
  0 & 1 & 0 & 0 \\
  0 & 1 & 1 & 1 \\
  \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
\end{array}$

Bot

$\begin{array}{ccccccc}
  r & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\
  1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\
  0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\
  0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 \\
  0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\
  1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\
  1 & 1 & 1 & 1 & 1 & 0 & 1 & 0 \\
  1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \\
  \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
\end{array}$
IKNP protocol

Bob has input $r \Rightarrow$ extend to matrix and secret share as $(T, T')$

Alice chooses random string $s$

OT for each column $\Rightarrow$ Alice obtains matrix $Q$
Bob has input $r \Rightarrow$ extend to matrix and secret share as $(T, T')$

Alice chooses random string $s$

OT for each **column** $\Rightarrow$ Alice obtains matrix $Q$
Bob has input $r$ ⇒ extend to matrix and secret share as $(T, T')$

Alice chooses random string $s$

OT for each **column** ⇒ Alice obtains matrix $Q$
IKNP protocol

Alice

Bob has input $r \Rightarrow$ extend to matrix and secret share as $(T, T')$

Alice chooses random string $s$

OT for each **column** ⇒ Alice obtains matrix $Q$
IKNP protocol

\[
\begin{array}{cccccc}
\hline
s = 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 \\
1 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\
1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\
0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\
1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 \\
1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 \\
0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\
\hline
\end{array}
\]

Alice

\[
\begin{array}{cccccc}
\hline
r & 1 & 1 & 1 & 0 & 1 & 1 & 0 \\
1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 \\
0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 \\
1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 \\
1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\
1 & 0 & 0 & 1 & 1 & 1 & 0 & 1 \\
\hline
\end{array}
\]

Bob

- Bob has input \( r \Rightarrow \) extend to matrix and secret share as \((T, T')\)
- Alice chooses random string \( s \)
- OT for each **column** ⇒ Alice obtains matrix \( Q \)
- Whenever \( r_i = 0 \), Alice row = Bob row
IKNP protocol

Bob has input $r \Rightarrow$ extend to matrix and secret share as $(T, T')$

Alice chooses random string $s$

OT for each column $\Rightarrow$ Alice obtains matrix $Q$

Whenever $r_i = 0$, Alice row = Bob row

Whenever $r_i = 1$, Alice row = Bob row $\oplus s$
IKNP protocol [IshaiKilianNissimPetrank03]

\[ q_i = \begin{cases} t_i & \text{if } r_i = 0 \\ t_{i+1} & \text{if } r_i = 1 \end{cases} \]

\[ t_1, t_2, t_3, \ldots \]

For every \( i \): Bob knows \( t_i \); Alice knows \( q_i \) and \( q_{i+1} \).

From Bob's perspective, he knows exactly one of Alice's two values: (Almost) an OT instance for each \( i \)!

Reusing \( s \) leads to linear correlations in OT strings.

Break correlations by applying random oracle:

\[ H(t_1, s), \ldots, H(t_n, s) \] pseudorandom given \( t_1, \ldots, t_n \) (secret \( s \)).

Random OT instance for each row, using base OT for each column.
IKNP protocol [IshaiKilianNissimPetrank03]

- For every $i$: Bob knows $t_i$; Alice knows $q_i$ and $q_i \oplus s$
For every $i$: Bob knows $t_i$; Alice knows $q_i$ and $q_i \oplus s$

From Bob’s perspective, he knows **exactly one** of Alice’s two values: (Almost) an OT instance for each $i$!
IKNP protocol

\[ q_i = \begin{cases} 
  t_i & \text{if } r_i = 0 \\
  t_i \oplus s & \text{if } r_i = 1 
\end{cases} \]

- For every \( i \): Bob knows \( t_i \); Alice knows \( q_i \) and \( q_i \oplus s \)
- From Bob’s perspective, he knows **exactly one** of Alice’s two values: (Almost) an OT instance for each \( i \!\)
  - Reusing \( s \) leads to linear correlations in OT strings
IKNP protocol [IshaiKilianNissimPetrunk03]

\[
q_i = \begin{cases} 
t_i & \text{if } r_i = 0 \\
t_i \oplus s & \text{if } r_i = 1 
\end{cases}
\]

\[
\begin{array}{cc}
H(t_1) & H(t_1 \oplus s) \\
H(t_2 \oplus s) & H(t_2) \\
H(t_3 \oplus s) & H(t_3) \\
\vdots & \vdots 
\end{array}
\]

\[
\begin{array}{c}
r_1 = 0 \\
r_2 = 1 \\
r_3 = 1 
\end{array}
\]

- For every \(i\): Bob knows \(t_i\); Alice knows \(q_i\) and \(q_i \oplus s\)
- From Bob’s perspective, he knows exactly one of Alice’s two values: (Almost) an OT instance for each \(i\)!
  - Reusing \(s\) leads to linear correlations in OT strings
- Break correlations by applying random oracle:
  - \(H(t_1 \oplus s), \ldots, H(t_n \oplus s)\) pseudorandom given \(t_1, \ldots, t_n\) (secret \(s\))
IKNP protocol

\[ q_i = \begin{cases} t_i & \text{if } r_i = 0 \\ t_i \oplus s & \text{if } r_i = 1 \end{cases} \]

For every \( i \): Bob knows \( t_i \); Alice knows \( q_i \) and \( q_i \oplus s \)

From Bob’s perspective, he knows **exactly one** of Alice’s two values: (Almost) an OT instance for each \( i \)!

- Reusing \( s \) leads to linear correlations in OT strings
- Break correlations by applying random oracle:
  - \( H(t_1 \oplus s), \ldots, H(t_n \oplus s) \) pseudorandom given \( t_1, \ldots, t_n \) (secret \( s \))

⇒ Random OT instance for each **row**, using **base OT** for each **column**
Tall matrices ($\lambda$ columns, $n \gg \lambda$ rows)
Tall matrices ($\lambda$ columns, $n \gg \lambda$ rows)

Base OTs by column
- $\lambda$ base OT instances
- transfer of $n$-bit strings
IKNP overview [IshaiKilianNissimPetrank03]

Tall matrices ($\lambda$ columns, $n \gg \lambda$ rows)

**Base OTs by column**
- $\lambda$ base OT instances
- transfer of $n$-bit strings

**Obtain extended OT instance by row**
- 1-2 evaluations of $H$ per row
Generalizing IKNP

IKNP says: “Bob has $r$”

KK13 says: \(0 7! 000; 1 7! 111\) is simple repetition code

Q: How do code properties (rate, distance) affect protocol?

[Reference: KolesnikovKumaresan13]
Generalizing IKNP

<table>
<thead>
<tr>
<th>$r$</th>
<th>1 1 1 1 1 1 1 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1 1 1 1 1 1 1 1</td>
</tr>
<tr>
<td>0</td>
<td>0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>0</td>
<td>0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>0</td>
<td>0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>1</td>
<td>1 1 1 1 1 1 1 1</td>
</tr>
<tr>
<td>0</td>
<td>0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>1</td>
<td>1 1 1 1 1 1 1 1</td>
</tr>
<tr>
<td>1</td>
<td>1 1 1 1 1 1 1 1</td>
</tr>
<tr>
<td>.</td>
<td>. . . . . . . .</td>
</tr>
</tbody>
</table>

- IKNP says: “Bob has $r \implies$ extend to a matrix”
Generalizing IKNP

IKNP says: “Bob has $r \Rightarrow$ extend to a matrix $\Rightarrow$ secret-share”
Generalizing IKNP

- IKNP says: “Bob has $r$ ⇒ extend to a matrix ⇒ secret-share”
- KK13 says: $0 \mapsto 000 \cdots; 1 \mapsto 111 \cdots$ is simple repetition code
Generalizing IKNP

- **IKNP** says: “Bob has \( r \Rightarrow \text{extend to a matrix} \Rightarrow \text{secret-share}”
- **KK13** says: \( 0 \mapsto 000 \cdots ; 1 \mapsto 111 \cdots \) is simple **repetition code**
- **Generalize** by using a different error-correcting code.

**Q:** How do code properties (rate, distance) affect protocol?
Coding view of IKNP:

- Bob has input $r$

```
\begin{array}{c}
r \\
1 \\
0 \\
0 \\
0 \\
. \\
. \\
. \\
\end{array}
```

Bob
Coding view of IKNP:

<table>
<thead>
<tr>
<th>$r$</th>
<th>$\cdots C(1) \cdots$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$\cdots C(0) \cdots$</td>
</tr>
<tr>
<td>0</td>
<td>$\cdots C(0) \cdots$</td>
</tr>
<tr>
<td>0</td>
<td>$\cdots C(0) \cdots$</td>
</tr>
<tr>
<td>0</td>
<td>$\cdots C(0) \cdots$</td>
</tr>
<tr>
<td>$\vdots$</td>
<td>$\vdots$</td>
</tr>
</tbody>
</table>

- Bob has input $r \Rightarrow$ **encode under** $C$

Bob
Coding view of IKNP:

- Bob has input \( r \Rightarrow \text{encode under } C \) and secret share as \((T, T')\)

<table>
<thead>
<tr>
<th>( r )</th>
<th>( \cdots t_1 \cdots )</th>
<th>( \cdots t_2 \cdots )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( \cdots t_1 \oplus C(1) \cdots )</td>
<td>( \cdots t_2 \oplus C(0) \cdots )</td>
</tr>
<tr>
<td>0</td>
<td>( \cdots t_3 \oplus C(0) \cdots )</td>
<td>( \cdots t_4 \oplus C(0) \cdots )</td>
</tr>
<tr>
<td>0</td>
<td>( \cdots )</td>
<td>( \cdots )</td>
</tr>
</tbody>
</table>
Bob has input $r$ ⇒ **encode under $C$** and secret share as $(T, T')$

OT for each **column** ⇒ Alice obtains matrix $Q$
Coding view of IKNP:

\[
\begin{array}{c}
\cdots q_1 \cdots \\
\cdots q_2 \cdots \\
\cdots q_3 \cdots \\
\cdots q_4 \cdots \\
\vdots
\end{array}
\]

\[
\begin{array}{c|c}
\hline
r & t_1 \cdots \\
1 & 0 \\
0 & \vdots \\
\hline
\end{array}
\]

\[
\begin{array}{c}
\cdots t_1 \oplus C(1) \cdots \\
\cdots t_2 \oplus C(0) \cdots \\
\cdots t_3 \oplus C(0) \cdots \\
\cdots t_4 \oplus C(0) \cdots \\
\vdots
\end{array}
\]

\[
t_i = q_i \oplus C(r_i) \land s
\]

- Bob has input \( r \) ⇒ **encode under** \( C \) and secret share as \((T, T')\)
- OT for each **column** ⇒ Alice obtains matrix \( Q \)
Codiﬁcation view of IKNP:

<table>
<thead>
<tr>
<th>q_1</th>
<th>q_2</th>
<th>q_3</th>
<th>q_4</th>
<th>\cdots</th>
</tr>
</thead>
</table>

\[
t_i = q_i \oplus C(r_i) \land s
\]

Alice

<table>
<thead>
<tr>
<th>r</th>
<th>t_1</th>
<th>t_2</th>
<th>t_3</th>
<th>t_4</th>
<th>\cdots</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>\cdots</td>
</tr>
</tbody>
</table>

Bob has input \( r \Rightarrow \text{encode under } C \) and secret share as \((T, T')\)

OT for each \textbf{column} \Rightarrow Alice obtains matrix \( Q \)

Sanity check (using repetition code):

- \( r_i = 0 \Rightarrow t_i = q_i \oplus (000 \cdots) \land s = q_i \)
- \( r_i = 1 \Rightarrow t_i = q_i \oplus (111 \cdots) \land s = q_i \oplus s \)
Coding view of IKNP:

\[ s, \{q_i\} \leftarrow \text{IKNP} \leftarrow r \]

\[ \{t_i\} \]

For every \( i \):
- Bob knows \( t_i \);
- Alice knows \( q_i \) and \( C(0)^s \)

\[ C(1)^s \]

\[ C(0)^s = 00 \]

▶ Rewrite from Bob’s point of view

▶ When \( C \) is a linear code:

\[ [C(a)^s][C(b)^s] = C(a \cdot b)^s \]

▶ Use random oracle to destroy correlations.
Coding view of IKNP:

- For every $i$: Bob knows $t_i$; Alice knows $q_i \oplus C(0) \land s$ and $q_i \oplus C(1) \land s$
Coding view of IKNP:

\[ t_i = q_i \oplus C(r_i) \land s \]

<table>
<thead>
<tr>
<th>( t_1 \oplus C(0) \land s \oplus C(0) \land s )</th>
<th>( t_1 \oplus C(0) \land s \oplus C(1) \land s )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t_2 \oplus C(1) \land s \oplus C(0) \land s )</td>
<td>( t_2 \oplus C(1) \land s \oplus C(1) \land s )</td>
</tr>
<tr>
<td>( t_3 \oplus C(1) \land s \oplus C(0) \land s )</td>
<td>( t_3 \oplus C(1) \land s \oplus C(1) \land s )</td>
</tr>
<tr>
<td>( \vdots )</td>
<td>( \vdots )</td>
</tr>
</tbody>
</table>

For every \( i \): Bob knows \( t_i \); Alice knows \( q_i \oplus C(0) \land s \) and \( q_i \oplus C(1) \land s \).

Rewrite from Bob’s point of view.
Coding view of IKNP:

$t_i = q_i \oplus C(r_i) \land s$

<table>
<thead>
<tr>
<th>$t_1 \oplus C(0 \oplus 0) \land s$</th>
<th>$t_1 \oplus C(0 \oplus 1) \land s$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_2 \oplus C(1 \oplus 0) \land s$</td>
<td>$t_2 \oplus C(1 \oplus 1) \land s$</td>
</tr>
<tr>
<td>$t_3 \oplus C(1 \oplus 0) \land s$</td>
<td>$t_3 \oplus C(1 \oplus 1) \land s$</td>
</tr>
<tr>
<td>\vdots</td>
<td>\vdots</td>
</tr>
</tbody>
</table>

$\{r_1 = 0, r_2 = 1, r_3 = 1\}$

$\{t_i\}$

- For every $i$: Bob knows $t_i$; Alice knows $q_i \oplus C(0) \land s$ and $q_i \oplus C(1) \land s$
- Rewrite from Bob’s point of view
- When $C$ is a **linear code**: $[C(a) \land s] \oplus [C(b) \land s] = C(a \oplus b) \land s$
Coding view of IKNP:

\[ t_i = q_i \oplus C(r_i) \land s \]

\[
\begin{array}{|c|c|}
\hline
  t_1 & t_1 \oplus C(1) \land s \\
\hline
  t_2 \oplus C(1) \land s & t_2 \\
\hline
  t_3 \oplus C(1) \land s & t_3 \\
\hline
  \vdots & \vdots \\
\hline
\end{array}
\]

- For every \( i \): Bob knows \( t_i \); Alice knows \( q_i \oplus C(0) \land s \) and \( q_i \oplus C(1) \land s \)
- Rewrite from Bob’s point of view
- When \( C \) is a **linear code**: \( [C(a) \land s] \oplus [C(b) \land s] = C(a \oplus b) \land s \) and \( C(0) \land s = 00 \cdots \)
Coding view of IKNP:

\[ t_i = q_i \oplus C(r_i) \land s \]

\[
\begin{array}{c|c}
H(t_1) & H(t_1 \oplus C(1) \land s) \\
H(t_2 \oplus C(1) \land s) & H(t_2) \\
H(t_3 \oplus C(1) \land s) & H(t_3) \\
\vdots & \vdots \\
\end{array}
\]

- For every \( i \): Bob knows \( t_i \); Alice knows \( q_i \oplus C(0) \land s \) and \( q_i \oplus C(1) \land s \)
- Rewrite from Bob’s point of view
- When \( C \) is a **linear code**: \([C(a) \land s] \oplus [C(b) \land s] = C(a \oplus b) \land s \) and \( C(0) \land s = 00 \cdots \)
- Use random oracle to destroy correlations
Generalizing IKNP:

Consider a code that encodes more bits $C : \{0, 1\}^3 \rightarrow \{0, 1\}^k$

\[ s, \{q_i\} \quad \text{IKNP} \quad r \quad \{t_i\} \]
Generalizing IKNP:

Consider a code that encodes more bits $C : \{0, 1\}^3 \rightarrow \{0, 1\}^k$

$$
\begin{array}{ccc}
q_1 \oplus C(000) \land s & \cdots & q_1 \oplus C(111) \land s \\
q_2 \oplus C(000) \land s & \cdots & q_2 \oplus C(111) \land s \\
q_3 \oplus C(000) \land s & \cdots & q_3 \oplus C(111) \land s \\
\vdots & \vdots & \vdots
\end{array}
$$

- For every $i$: Alice can compute (8 things)

$$q_i \oplus C(000) \land s, \quad q_i \oplus C(001) \land s, \quad \ldots \quad q_i \oplus C(111) \land s$$
Generalizing IKNP:

Consider a code that encodes more bits $C : \{0, 1\}^3 \rightarrow \{0, 1\}^k$

$$t_i = q_i \oplus C(r_i) \land s$$

For every $i$: Alice can compute (8 things)

$$q_i \oplus C(000) \land s, \quad q_i \oplus C(001) \land s, \quad \ldots \quad q_i \oplus C(111) \land s$$
Generalizing IKNP:

Consider a code that encodes more bits $C : \{0, 1\}^3 \rightarrow \{0, 1\}^k$

$$t_i = q_i \oplus C(r_i) \land s$$

<table>
<thead>
<tr>
<th>$t_1 \oplus C(r_1 \oplus 000) \land s$</th>
<th>$\cdots$</th>
<th>$t_1 \oplus C(r_1 \oplus 111) \land s$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_2 \oplus C(r_2 \oplus 000) \land s$</td>
<td>$\cdots$</td>
<td>$t_2 \oplus C(r_2 \oplus 111) \land s$</td>
</tr>
<tr>
<td>$t_3 \oplus C(r_3 \oplus 111) \land s$</td>
<td>$\cdots$</td>
<td>$t_3 \oplus C(r_3 \oplus 111) \land s$</td>
</tr>
<tr>
<td>$\vdots$</td>
<td>$\vdots$</td>
<td>$\vdots$</td>
</tr>
</tbody>
</table>

For every $i$: Alice can compute (8 things)

$$q_i \oplus C(000) \land s, \quad q_i \oplus C(001) \land s, \quad \ldots \quad q_i \oplus C(111) \land s$$

Bob knows exactly 1 of the 8 values (corresponding to $r_i$)

- Others are of the form $t \oplus c \land s$ for known $t$ and **codeword** $c$
Generalizing IKNP:

Consider a code that encodes more bits \( C : \{0, 1\}^3 \rightarrow \{0, 1\}^k \)

\[
\begin{array}{ccc}
H(t_1 \oplus C(r_1 \oplus 000) \land s) & \cdots & H(t_1 \oplus C(r_1 \oplus 111) \land s) \\
H(t_2 \oplus C(r_2 \oplus 000) \land s) & \cdots & H(t_2 \oplus C(r_2 \oplus 111) \land s) \\
H(t_3 \oplus C(r_3 \oplus 000) \land s) & \cdots & H(t_3 \oplus C(r_3 \oplus 111) \land s) \\
\vdots & \vdots & \vdots \\
\end{array}
\]

\[
t_i = q_i \oplus C(r_i) \land s
\]

- For every \( i \): Alice can compute (8 things)
  \[
  q_i \oplus C(000) \land s, \quad q_i \oplus C(001) \land s, \quad \ldots \quad q_i \oplus C(111) \land s
  \]

- Bob knows exactly 1 of the 8 values (corresponding to \( r_i \))
  - Others are of the form \( t \oplus c \land s \) for known \( t \) and codeword \( c \)

- In the random oracle model:
  - \( H(t_1 \oplus c_1 \land s), \ldots H(t_n \oplus c_n \land s) \) pseudorandom if all \( c_i \) have Hamming weight \( \geq \lambda \)
Generalizing IKNP:

Using a code $C : \{0, 1\}^\ell \rightarrow \{0, 1\}^k$ with minimum distance $\lambda$ gives you 1-out-of-$2^\ell$ OT extension (from $k$ base OTs)

[KolesnikovKumaresan13]:
- Walsh-Hadamard code $C : \{0, 1\}^8 \rightarrow \{0, 1\}^{256}$ (min. dist. 128)
- 1-out-of-256 OT

[OrruOrsiniScholl16]:
- BCH code $C : \{0, 1\}^{76} \rightarrow \{0, 1\}^{512}$ (min. dist. 171)
- 1-out-of-$2^{76}$ OT

[KolesnikovKumaresanRosulekTrieu16]:
- Any pseudorandom function $C : \{0, 1\}^* \rightarrow \{0, 1\}^{\sim 480}$
- Linearity and decoding properties not needed (only min. dist. whp!)
- 1-out-of-$\infty$ OT
### Perspective

<table>
<thead>
<tr>
<th></th>
<th>semi-honest</th>
<th>malicious</th>
<th>semi-honest</th>
<th>malicious</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-out-of-2</td>
<td>28 million / sec</td>
<td>24 million / sec</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1-out-of-N</td>
<td>2.5 million / sec</td>
<td>1.8 million / sec</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

OTs are cheap!
Perspective

<table>
<thead>
<tr>
<th></th>
<th>semi-honest</th>
<th>malicious</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-out-of-2</td>
<td>28 million / sec</td>
<td>24 million / sec</td>
</tr>
<tr>
<td>1-out-of-N</td>
<td>2.5 million / sec</td>
<td>1.8 million / sec</td>
</tr>
</tbody>
</table>

OTs are cheap!