1 Oblivious Transfer Cont.

1.1 Pre-processing Oblivious Transfer

For the pre-processing OT protocol, one round of OT is used to establish keys. Thereafter, an arbitrary number of OT’s over different messages can be performed for using these keys.

\[ k_0, k_1 \leftarrow \{0, 1\}^n \]

\[ c \leftarrow \{0, 1\} \]

First phase

\[ k_0, k_1 \]

\[ c \]

Second phase

\[ m_0, m_1 \]

\[ b \]

if \( z = 0 \)

\[ y_0 = m_0 \oplus k_0 \]

\[ y_1 = m_1 \oplus k_1 \]

else

\[ y_0 = m_0 \oplus k_1 \]

\[ y_1 = m_1 \oplus k_0 \]

\[ y_0, y_1 \]

\[ b \oplus c \]

\[ R \text{ learns } y_0 \oplus k_c = m_b \]

Figure 1: Protocol for Pre-processing OT
1.2 OT Extension

An OT extension protocol turns \(k\) OTs on \(m\)-bit strings into \(m\) OTs on \(n\)-bit strings, where \(k\) is the security parameter.

Sender\(((x_{1,0}, x_{1,1}), \ldots, (x_{m,0}, x_{m,1}))\)

\[ s \leftarrow \{0, 1\}^k \]

run \(k\) OTs as receiver
using \(s_1, \ldots, s_k\) as inputs
S learns \(Q \in \{0, 1\}^{m \times k}\)

\[
\begin{pmatrix}
Q_1 & \cdots & Q_k
\end{pmatrix}
\]

Let \(Q^i\) be the \(i^{th}\) row of \(Q\)

\[
\mathcal{H}(R^i) \oplus x_{i,0} \quad \mathcal{H}(Q^i \oplus s) \oplus x_{i,1}
\]

Receiver\((r_1, \ldots, r_m)\)

\[ T \leftarrow \{0, 1\}^{m \times k} \]

run \(k\) OTs as sender
using \((T_j, T_j \oplus r)\) as inputs

\[
\begin{pmatrix}
T_1 & T_1 \oplus r & \cdots & T_k & T_k \oplus r
\end{pmatrix}
\]

R recovers \(x_{1,r_1}, \ldots, x_{m,r_m}\)
if \(r_i = 0\), R knows \(Q^i = T^i\)
if \(r_i = 1\), R knows \(Q^i \oplus s = T^i\)

Figure 2: Protocol for OT extension

1.2.1 Assumption on Randomness of the Hash Function

For arbitrary \(T^1, \ldots, T^m\) and \(s\), the hash function outputs, \(\mathcal{H}(s \oplus T^1), \mathcal{H}(s \oplus T^2), \ldots, \mathcal{H}(s \oplus T^m)\), should be indistinguishable from uniform random, even given \(T^1, \ldots, T^m\).
1.3 GMW (Goldreich-Micali,Wigderson) Approach to semi-honest two-party computation

Secure computation of arbitrary circuits from OT. Assume we have a Boolean circuit with \(2\ell\) inputs, the first half are from \(P_1\), the second half are from \(P_2\). The gates may have arbitrary fan-in and fan-out. At the bottom we have some number of output gates, and both parties learn all the outputs.

The approach is to have 2-out-of-2 secret sharing for every wire value. The protocol proceeds layer by layer, beginning with the input layer.
1.3.1 Input Layer

<table>
<thead>
<tr>
<th>P₁(x₁,...,xₖ)</th>
<th>P₂(y₁,...,yᵣ)</th>
</tr>
</thead>
<tbody>
<tr>
<td>choose s₁,...,sₖ ∈ {0,1}</td>
<td>choose r₁,...,rᵣ ∈ {0,1}</td>
</tr>
<tr>
<td>s₁,...,sₖ</td>
<td>r₁,...,rᵣ</td>
</tr>
<tr>
<td>P₁ has P₁’s input wires, and x₁ ⊕ s₁,...,xₖ ⊕ sₖ (shares of P₂’s inputs)</td>
<td>P₂ has P₂’s input wires, and y₁ ⊕ r₁,...,yₖ ⊕ rₖ (shares of P₁’s inputs)</td>
</tr>
</tbody>
</table>

1.3.2 XOR Gate

No communications are required for an XOR gate - each party can construct the shares of the output using their existing shares of the inputs.

| γ₃ = γ₁ ⊕ γ₂ |
| r₁ ⊕ s₁ = γ₁ |
| r₂ ⊕ s₂ = γ₂ |
| P₁ has r₁, r₂ |
| P₂ has s₁, s₂ |
| define r₃ = r₁ ⊕ r₂ |
| define s₃ = s₁ ⊕ s₂ |

1.3.3 NOT Gate

NOT gates are easy - just agree that one player (e.g., P₁) flips the bit.
1.3.4 AND Gate

Each AND gate requires an invocation of OT.

\[
\begin{align*}
\gamma_3 &= \gamma_1 \land \gamma_2 \\
r_1 \oplus s_1 &= \gamma_1 \\
r_2 \oplus s_2 &= \gamma_2
\end{align*}
\]

P₁ has \(r₁, r₂\)

choose \(r₃ \leftarrow \{0, 1\}\)

this is P₁’s share of \(\gamma₃\)

P₂ has \(s₁, s₂\)

use 1-out-of-4 OT to select appropriate row from table

<table>
<thead>
<tr>
<th>s₁</th>
<th>s₂</th>
<th>s₃</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>((r₁ \land r₂) \oplus r₃)</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>((r₁ \land \neg r₂) \oplus r₃)</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>((\neg r₁ \land r₂) \oplus r₃)</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>((\neg r₁ \land \neg r₂) \oplus r₃)</td>
</tr>
</tbody>
</table>