Lecture 39: GMW Protocol
Recall

- Last lecture we saw that we can securely compute any function using oblivious transfer (which can be constructed from the RSA assumption).
- However, the protocol is efficient only when the function has constant size.
Today we shall learn the Goldreich-Micali-Wigderson (GMW) Protocol to securely compute any function that can be efficiently computed.
Recall: Additive Secret Sharing Scheme I

- Let \((G, \circ)\) be a group
- For any \(s \in G\), we pick \(s_A \leftarrow G\) and define \(s_B = \text{inv}(s_A) \circ s\)
- Note that just given \(s_A\), the secret \(s\) is perfectly hidden
- Note that just given \(s_B\), the secret \(s\) is perfectly hidden
- But, given both \(s_A\) and \(s_B\) we can reconstruct \(s\)
- This secret sharing scheme shall be referred to as the “additive secret sharing scheme” (you have already seen this scheme in the midterm)
An Example.

- Consider the group $\langle \{0, 1\}, \oplus \rangle$, where $\oplus$ is the bit-xor.
- Then the additive secret shares of a secret bit $s$ is $s_A \leftarrow \{0, 1\}$ and $s_B = s_A \oplus s$.
- Note that the secret $s = s_A \oplus s_B$. 
Basic Step 1

- Suppose we have two wires $u$ and $v$
- The values of these two wires in a circuit be $\text{val}(u)$ and $\text{val}(v)$
- Suppose the secret shares of $\text{val}(u)$ be $\text{val}(u)_A$ and $\text{val}(u)_B$
- Suppose the secret shares of $\text{val}(v)$ be $\text{val}(v)_A$ and $\text{val}(v)_B$
- Let $G$ be a gate where wire $u$ and $v$ are inputs and wire $w$ is the output. For example, the gate $G$ can be the AND-gate, NAND-gate, XOR-gate, etc.
- So, the value of the wire $w$ is $\text{val}(w) = G(\text{val}(u), \text{val}(v))$

\[ \begin{array}{c}
\text{val}(u) \rightarrow \quad G \\
\text{val}(v) \rightarrow \quad \text{val}(w)
\end{array} \]
Basic Step II

Suppose

- Alice already has $\text{val}(u)_A$ and $\text{val}(v)_A$
- Bob already has $\text{val}(u)_B$ and $\text{val}(v)_B$
- Alice samples $\text{val}(w)_A \leftarrow \{0, 1\}$

What is the share $\text{val}(w)_B$?

$$\text{val}(w)_B = \text{val}(w)_A \oplus G\left(\text{val}(u)_A \oplus \text{val}(u)_B, \text{val}(v)_A \oplus \text{val}(v)_B\right)$$

So, the value $\text{val}(w)_B$ is a function of 3-bit input from Alice and 2-bit input from Bob, i.e., it is a function of constant size. Now, we can efficiently and securely compute this function!
Suppose Alice has private input $x = (x_1, x_2, \ldots, x_n)$

Suppose Bob has private input $y = (y_1, y_2, \ldots, y_n)$

Alice and Bob are interested in computing a function that is described by a circuit $C$. The output of the circuit is $z = C(x, y)$
The GMW Protocol II

**Base Case.** Additively secret sharing the input wires

- Suppose the wires \(\{1, 2, \ldots, n\}\) correspond to Alice’s input \((x_1, x_2, \ldots, x_n)\), respectively. Alice picks random \(\text{val}(i)_A \leftarrow \{0, 1\}\), for \(i \in \{1, 2, \ldots, n\}\). Alice sends \(\text{val}(i)_B = x_i \oplus \text{val}(i)_A\) to Bob.

- Suppose the wires \(\{n + 1, n + 2, \ldots, 2n\}\) correspond to Bob’s input \((y_1, y_2, \ldots, y_n)\), respectively. Bob picks random \(\text{val}(n + i)_B \leftarrow \{0, 1\}\), for \(i \in \{1, 2, \ldots, n\}\). Bob sends \(\text{val}(n + i)_A = y_i \oplus \text{val}(n + i)_B\) to Alice.
Inductively Computing Internal Wires. Suppose Alice and Bob want to securely compute the output of a gate $G$ whose input wires are $u$ and $v$, and the output wire is $w$. Assume, by induction hypothesis, that $\text{val}(u)_A$ and $\text{val}(v)_A$ are with Alice, and $\text{val}(u)_B$ and $\text{val}(v)_B$ are with Bob.

- First, Alice picks $\text{val}(w)_A \leftarrow \{0, 1\}$
- Next, Alice and Bob securely compute the function that outputs the following value to Bob (we already know how to do this)

$$
\text{val}(w)_B = \text{val}(w)_A \oplus G \left( \text{val}(u)_A \oplus \text{val}(u)_B, \text{val}(v)_A \oplus \text{val}(v)_B \right)
$$

Repeat this for all gates.
Finalizing the Output. Suppose the output wires are \{s + 1, s + 2, \ldots, s + m\}. Alice has the values val(s + i)_A and Bob has the values val(s + i)_B, for i ∈ \{1, 2, \ldots, m\}.

- Alice and Bob exchange the values val(s + i)_A and val(s + i)_B, for i ∈ \{1, 2, \ldots, m\}, to reconstruct val(s + i).

This is the output \(z = (\text{val}(s + 1), \text{val}(s + 2), \ldots, \text{val}(s + m))\)

So, we can securely evaluate any circuit in time proportional to its size!
Consider the following example understand how the GMW-protocol can be helpful

- Consider the example of Dutch flower auction
- Suppose Alice has an $n$-bit bid that is even, and Bob has an $n$-bit bid that is odd
- So, each party has $2^{n-1}$ possible inputs (bids)
- If we securely evaluate this function using the approach introduced in the previous class, then we need $2^n$ rounds, which is inefficient
How do we securely perform this task using the GMW-protocol?

- Write an efficient circuit that evaluates the maximum of the two inputs \((x_1, \ldots, x_n)\) and \((y_1, \ldots, y_n)\) (What is the smallest circuit that you can design?)

- Use RSA-based \(m\)-choose-1 OT protocol to securely compute this circuit using the GMW-protocol
What are the tradeoffs between these two protocols?

- The first protocol is perfectly secure, while the second protocol is secure only against computationally bounded parties.
- The first protocol is inefficient, while the second protocol is efficient.