CSC 2426: Fundamentals of Cryptography **Fall 2023**

Lecture 10: Oblivious Transfer and Secure Two-party Computation

Instructor: Akshayaram Srinivasan *Scribe: Yuntao Cai*

Date: 27 November, 2023

Disclaimer: *These notes have not been subjected to the usual scrutiny reserved for formal publications. They may be distributed outside this class only with the permission of the Instructor.*

10.1 Introduction

Last class we saw DDH - Assumption. A Recap: Given a cyclic group (G, $\cdot$) generated by generator g $\in$ G.

$$G = \{g^0, ...g^{p-1}\}$$

Any element g^x can be mapped to $x \in \mathbb{Z}_p = \{0, 1, ...p-1\}$ There are two operations that are easy to compute.

1) $h, h^{'} \in G$, one can compute $h * h^{'}$ in polynomial time (in the description size of h)
2) $h \in G$ and $x \in \mathbb{Z}_p$, $h^x \in G$ can be computed in polynomial time.

The Decisional Diffie-Hellman Assumption (DDH) states that:

$$\{G, p, g, g^x, g^y, g^{xy}\}_{x,y \leftarrow \mathbb{Z}_p} \approx_c \{G, p, g, g^x, g^y, g^z\}_{x,y,z \leftarrow \mathbb{Z}_p}$$

In the last class, we saw how to construct a public-key encryption scheme from DDH assumption. Let us recall the construction here.
The key generation algorithm outputs $pk = g^x, sk = x$ (where $x \leftarrow \mathbb{Z}_p$). To encrypt a message $m \in \mathbb{G}$, we sample $r \leftarrow \mathbb{Z}_p$, $c = (g^r, g^{x \cdot r} \cdot m)$. The view of the adversary is given by:

$$\{G, p, g, g^x, g^r, g^{x \cdot r} \cdot m)\}_{x,r \leftarrow \mathbb{Z}_p} \approx_c \{G, p, g, g^x, g^r, g^z\}_{x,r,z \leftarrow \mathbb{Z}_p} \text{ (from the DDH assumption)}$$

To encrypt a single bit message, encode $m = 0 \implies g^0$, $m = 1 \implies g^1$

10.2 Oblivious Transfer

In this lecture, we will look at another key cryptographic primitive called Oblivious Transfer. Oblivious transfer is a protocol that is run between two parties: a receiver and a sender.

	Receiver	*Sender*
Input	$b \in \{0, 1\}$	$(m_0, m_1) \in \{0, 1\}^2$
Output	m_b	no output

At the end of the interaction, receiver will have learned content of m_b.

Consider the trivial example where sender sends (m_0, m_1) to receiver. This reveals m_{1-b} to receiver.

Consider another example. Receiver sends b to sender, and sender sends m_b. This reveals b to sender.

We want receiver to learn m_b, not m_{1-b}, and sender does not learn b. This protects both receiver and sender.

Correctness: At the end of protocol, receiver learns m_b.

Semi-honest security: By semi-honest security, we mean that the parties follow the protocol but try to learn additional information about the private inputs of the other parties from their view.

- Receiver security: $\forall(m_0, m_1)$:
$$View_S\langle R(0), S(m_0, m_1)\rangle \approx_c View_S\langle R(1), S(m_0, m_1)\rangle$$
- Sender security: $\forall b \in \{0,1\}, \forall(m_0, m_1) \in \{0,1\}^2$:
$$View_R\langle R(b), S(m_0, m_1)\rangle \approx_c View_R\langle R(b), S(m_b, m_b)\rangle$$

Here, $View$ of a party includes its private input, the private random coin tosses, and the messages received from the other party.

We now show that a simple correlation called random OT correlation is sufficient to realize an OT protocol between the sender and the receiver on any input. In the random OT correlation, the sender receives two random bits (s_0, s_1) and the receiver obtains a random bit r along with s_r. We now show how to realize OT on any private input using such random OT correlation.

$$\begin{array}{lrl}
 & Receiver & Sender \\
Input: & b \in \{0,1\} & (m_0, m_1) \in \{0,1\}^2 \\
Correlation: & (r, s_r), r \leftarrow \{0,1\} & (s_0, s_1) \in \{0,1\}^2 \\
 & \xrightarrow{b \oplus r} & \\
 & \xleftarrow{c_0 = m_0 \oplus s_{b\oplus r}, c_1 = m_1 \oplus s_{1\oplus b\oplus r}} & \\
m_b = c_b \oplus s_r & &
\end{array}$$

It is easy to see that the above protocol is information-theoretically secure if r and s_0, s_1 are randomly chosen. We now show that how to build oblivious transfer protocol from the DDH assumption.

The main intuition behind this construction is the following. The receiver samples two public keys pk_0 and pk_1 for the public-key encryption described above. These public keys are sampled such that the receiver knows the secret-key for pk_b where b is its private input but it does not know any information about the secret key for pk_{1-b}. If that was the case, the sender can send an encryption of m_0 under pk_0 and m_1

under pk_1. The receiver can use the secret key for pk_b to recover m_b. Further, we can use the security of the encryption scheme to switch the encrypted message under pk_{1-b} from m_{1-b} to m_b. However, the key challenge here is how to ensure that receiver samples pk_{1-b} without knowing the corresponding secret key. Consider the following interaction.

$$\begin{array}{rcl}
\mathit{Receiver} & & \mathit{Sender} \\
 & & x \leftarrow \mathbb{Z}_p \\
 & \xleftarrow{g^x} & \\
y \leftarrow \mathbb{Z}_p & & \\
pk_b = g^y, pk_{1-b} = g^{x-y} & & \\
 & \xrightarrow{(pk_0, pk_1)} & \\
 & \xleftarrow{c_0 \leftarrow Enc(pk_0, m_0),\, c_1 \leftarrow Enc(pk_1, m_1)} & \\
Dec(y, c_b) = m_b & &
\end{array}$$

Correctness: trivial.

Receiver security:

$$\begin{aligned}
& pk_0 \equiv pk_1 \\
& b = 0 : (pk_0, pk_1) = (g^y, g^{x-y}) \\
& b = 1 : (pk_0, pk_1) = (g^{x-y}, g^y) \\
& \text{Let } x - y = y' \\
& b = 1 : (pk_0, pk_1) = (g^{y'}, g^{x-y'})
\end{aligned}$$

Since y is randomly distributed, $x - y$ is randomly distributed. Hence, y' is identically distributed to y. Thus, the two public-keys when $b = 0$ and when $b = 1$ are identically distributed. Therefore, the receiver security holds unconditionally.

Sender security: We will prove sender security assuming the DDH assumption. Suppose the sender security does not hold. That is, there exists a distinguisher that can distinguish between

$$View_R\langle R(b), S(m_0, m_1)\rangle \text{ and } View_R\langle R(b), S(m_b, m_b)\rangle$$

with non-negligible advantage. We will use this distinguisher to break the security of the encryption scheme. Specifically, we give a reduction that contradicts the security of the encryption scheme. The reduction obtains b (which is receiver's private input) and m_0, m_1 which are sender's private inputs. The reduction chooses a uniform random tape for the receiver. Note that this random tape includes the coins for sampling y. By setting the random tape, the reduction knows y that the receiver samples.

$$\begin{array}{rcccr}
\textit{Receiver} & & \textit{Reduction} & & \textit{Challenger} \\
 & & \xleftarrow{pk \cdot g^y} & & \xleftarrow{pk} \\
pk_b = g^y, pk_{1-b} = pk & & & & \\
 & \xrightarrow{(pk_0, pk_1)} & & & \xrightarrow{m_b, m_{1-b}} \\
 & & & & \xleftarrow{c^*} \\
 & & c_b = Enc(p_{kb}, m_b), c_{1-b} = c^* & & \\
 & \xleftarrow{(c_0, c_1)} & & &
\end{array}$$

The reduction generates the messages in the protocol as described above and generate the view of the receiver. It runs the distinguisher on this view and outputs whatever it outputs. If c^* is an encryption of m_{1-b}, then the input to the distinguisher is identically distributed to $View_R\langle R(b), S(m_0, m_1)\rangle$. Else, it is distributed identically to $View_R\langle R(b), S(m_b, m_b)\rangle$. Since we assumed that the distinguisher distinguishes between the above two views with non-negligible advantage, the reduction breaks the security of the encryption scheme which is a contradiction. This completes the proof of security.

10.3 Secure Two-party Computation

In this section, we are going to see how to use oblivious transfer to let two parties compute an arbitrary circuit on their private inputs while only leaking the output and nothing else.

Consider two parties: P_0 and P_1. P_0 has a private input $x \in \{0,1\}^n$ and P_1 has a private input $y \in \{0,1\}^n$. The parties have a circuit C (comprises of AND and XOR gates) as common input. The parties wish to compute the output of C on (x, y).

This task is trivial if the parties exchange their inputs in the clear with each other. That is, the party P_0 can send x to P_1 and P_1 can in turn send y to P_0 and the parties compute the output of the circuit locally. However, in this approach, the private inputs of each party is revealed to the other in the clear. Is there a way to ensure that the parties only learn the output of the circuit on their private inputs without leaking any other information? This is possible using secure computation.

Toy Example-1: AND gate. Suppose the parties want to compute an AND gate on their private inputs. That is, the party P_0 has a bit a and party P_1 has a bit b and the parties wish to learn $a \cdot b$. We can accomplish this using an OT protocol. Specifically, P_0 plays the role of the receiver with its private input set to a and P_1 plays the role of the sender with its two inputs bits set to $(0 \cdot b, 1 \cdot b)$. The output of OT reveals $a \cdot b$ to P_0 and it can send this result back to P_1.

Toy Example-2: AND-XOR gate. Suppose P_0's private input is a bit a and P_1's private inputs are two bits (b, r). The parties wish to compute $a \cdot b \oplus r$. We will again use an OT protocol to accomplish this. P_0 plays the role of the receiver with its private input set to a and P_1 plays the role of the sender with its two inputs bits set to $(0 \cdot b \oplus r, 1 \cdot b \oplus r)$. The output of OT reveals $a \cdot b \oplus r$ to P_0 and it can send this result back to P_1. If P_0 does not send back the result, then P_0 holds $a \cdot b \oplus r$ and P_1 holds r. If r is randomly sampled, then neither party knows any information about $a \cdot b$ but using $a \cdot b \oplus r$ and r, they can recover $a \cdot b$.

We will now show how to use the above simple example to securely compute every circuit. The protocol proceeds as follows:

- **Input Sharing Phase.** For each $i \in [n]$, P_0 chooses a random bit r_i and P_1 chooses a random bit s_i. P_0 sends $r_1, \ldots, r_n$ to P_1 and P_1 sends $s_1, \ldots, s_n$ to P_0. The parties locally compute $r_i \oplus x_i$ and $s_i \oplus y_i$. Note that for every input wire P_1 and P_0 holds two bits such that the XOR of these two bits give the value carried by that wire. This is called as an additive secret sharing of the wire values. Note that each party learns no information about the value carried by the wire belonging to the other party.

- **Computing the Circuit.** For every wire in the circuit, we will maintain the invariant that the parties will compute an additive sharing of the wire value and each party learns no information about the value carried by this wire unless it is an input wire belonging to this party. For the input wires, the invariant holds after the input sharing phase. This is the base case and we will ensure that the invariant holds using induction on the levels of the circuit. Assume that invariant holds for each wire up to level $i-1$ from the input level. Let g be an arbitrary gate in level-i. Let α and β be the values carried by the input wires to this gate. Since these two wires belong to lower levels, by induction hypothesis, the parties P_0 and P_1 hold (α_0, β_0) and (α_1, β_1) respectively such that $\alpha_0 \oplus \alpha_1 = \alpha$ and $\beta_0 \oplus \beta_1 = \beta$ and the parties have no information about α and β respectively (in a computational sense). If g was an XOR gate, then the parties maintain the invariant by setting the shares of the output wire value of g to be $\alpha_0 \oplus \beta_0$ and $\alpha_1 \oplus \beta_1$ respectively. If g was an AND gate, then the goal is to have an additive secret sharing of $\alpha \cdot \beta = (\alpha_0 \oplus \alpha_1) \cdot (\beta_0 \oplus \beta_1) = \alpha_0 \cdot \beta_0 \oplus \alpha_0 \cdot \beta_1 \oplus \alpha_1 \cdot \beta_0 \oplus \alpha_1 \cdot \beta_1$. Note that if the parties generate an additive secret sharing of each of the four terms, then we are done. For the two outer terms, the party P_i can set its share to be $\alpha_i \cdot \beta_i$ and the other party can set its share to be 0. To compute shares of each of the two inner terms, one party (say, P_0) chooses random bits γ_1, γ_2 and uses Toy example-2 to reveal $\alpha_0 \cdot \beta_1 \oplus r_1$ and $\alpha_1 \cdot \beta_0 \oplus \gamma_2$ to P_1. This corresponds to an additive sharing of two inner terms.

- **Reconstructing the Output:** The parties exchange the shares of the output wire to reconstruct the outpu by XORing the shares together.