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

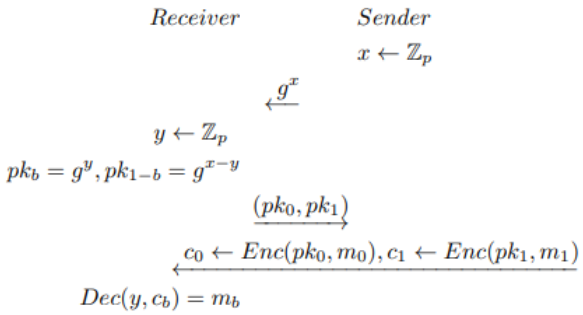
Lecture 11: Secure Multiparty Computation

Instructor: Akshayaram Srinivasan *Scribe: Vedic Sharma*

Date: December 5, 2023

11.1 Previous Class Review

In the previous class we covered oblivious transfer (OT) between a sender and a receiver. The main idea was as follows. Suppose a sender has two messages m_0 and m_1. The receiver has a bit $b \in \{0,1\}$. How can the receiver (without sharing b) learn the value of m_b without learning m_{1-b}? The solution known as the OT protocol uses a DDH assumption.



This can be used to construct a secure two-party computation. In such a scenario, party P_0 and P_1 each having a private input $x_0, x_1 \in \{0,1\}$ respectively wish to compute an arbitrary circuit C. We assumed that the circuit consists of purely XOR and AND gates. Each party keeps track of a value for each of the wires in the circuit. The invariant is the additive secret sharing of the wire values is equal to the actual value of the wire. Parties share using OT and compute their own wire values at each level while maintaining the invariant. On the final output wire, parties share their values to determine the computation of the circuit.

11.2 Secure MultiParty Computation Using OT

We have seen how two parties can compute an arbitrary circuit on their private inputs while only leaking the output and nothing else. In this section, we will extend this idea to multiple parties. Consider n parties: $P_1, P_2, ..., P_n$ with private inputs $x_1, x_2, ..., x_n \in \{0,1\}^n$. The parties want to compute the function $f(x_1, x_2, ..., x_n)$. Here f is a function that is composed of only two operations (AND and XOR). Similar to the two party assumption, the output of f is a single public value. We require that the method we construct should both follow:

Correctness: Each party learns the true value of $f(x_1, x_2, ..., x_n)$

Privacy: Each party does not learn the private inputs of other parties. For example P_1 does not learn $x_2, x_3, ..., x_n$.

11.2.1 Input Sharing Phase

Each party P_i splits their private input x_i as follows. They randomly sample $n-1$ values $x_{i,1}, x_{i,2}, ..., x_{i,n-1}$ from $\{0, 1\}$. Then they set

$$x_{i,n} = x_i \oplus x_{i,1} \oplus x_{i,2} \oplus ... \oplus x_{i,n-1}$$

Notice that x_i can be computed if a party has $x_{i,j}$ for all $j = 1, 2, ..., n$, but cannot be computed with even one less value. A party P_i shares $x_{i,j}$ to party P_j.

11.2.2 Maintaining Wire Values and Invariant

The function can be broken down to a electric circuit. Thus computing the function is equivalent to applying multiple different AND or XOR operations through wires. The value after a particular operation can be visualized as a wire and will be denoted by w.

During each step of the computation, each party P_i maintains a wire value w_i for each wire. This value may not be w, but these values follow an important invariant. That is,

$$w = w_1 \oplus w_2 \oplus ... \oplus w_n$$

Further, they satisfy that for any subset S of size $n-1$, $\{w_i\}_{i \in S}$ provides no information about w. The idea is that once all the operations in f are performed, parties can share their wire values (of the output wire) to determine the function value.

11.2.3 Computing the Function Value

We will show that parties determine their wire values using a recursive method. As there are two gates, we split this into two cases. One where the gate is XOR and the other where the gate is AND.

Initial Conditions For a input wire w for an arbitrary party P_i, the wire values are given by $w_j = x_{i,j}$. From the way we constructed these values, observe that the invariant is preserved.

Case 1: The gate is XOR gate.
Suppose the gate takes input α and β and computes $\alpha \oplus \beta$. How can the parties determine their wire values? In this case suppose α_i and β_i are the corresponding wire value for party P_i. Then consider the following equations.

$$\alpha = \alpha_1 \oplus \alpha_2 \oplus ... \oplus \alpha_n$$

$$\beta = \beta_1 \oplus \beta_2 \oplus ... \oplus \beta_n$$

$$\alpha \oplus \beta = (\alpha_1 \oplus \beta_1) \oplus (\alpha_2 \oplus \beta_2) \oplus ... \oplus (\alpha_n \oplus \beta_n)$$

A party P_i can do the operation $\alpha_i \oplus \beta_i$ locally to compute the value of the wire corresponding to $\alpha \oplus \beta$. As we see from the operation above, this is enough to retain the invariant.

Case 2: The gate is AND gate.
Now let's suppose that the gate in question is an AND gate. Thus

$$\begin{aligned}\alpha\beta &= (\alpha_1 \oplus \alpha_2 \oplus ... \oplus \alpha_n)(\beta_1 \oplus \beta_2 \oplus ... \oplus \beta_n) \\ &= \sum_i \left(\alpha_i\beta_i \oplus \sum_{j \neq i} \alpha_i\beta_j \right)\end{aligned}$$

where the $\sum$ represents the addition over modulo 2 (i.e. XOR operation). Note that it is sufficient for the parties to compute an XOR secret sharing of each of the terms in the above equation. Since α_i and β_i are available to P_i, it sets its share to be $\alpha_i \cdot \beta_i$ and the rest of the parties set their shares to be 0. α_i is available with P_i and β_j is available with P_j. These two parties use the OT-based protocol from the previous lecture to compute an XOR secret secret sharing of $\alpha_i \cdot \beta_j$. The rest of the parties set their share of this term to be 0. This ensures that the parties hold an XOR secret sharing of $\alpha\beta$ such that any subset of $n-1$ parties learn no information about $\alpha\beta$.

11.2.4 Reconstructing the Output

The parties exchange their output wire value with all the other parties. Then each party simply applies the XOR operation on all of them to get the desired function value.

11.3 Shamir's Secret Sharing

Shamir's secret sharing or threshold secret sharing is an efficient secret sharing scheme. To understand how the algorithm works we first what (t, n) secret sharing is, then discuss polynomial interpolation, and finally Shamir's protocol.

11.3.1 t-out-of-n Secret Sharing

A t-out-of-n secret sharing scheme is a pair of PPT algorithms (Share, Recon) having the following properties

- Share(x) outputs $(s_1, s_2, ..., s_n)$.
- Recon$(\{s_i\}_{i \in T})$ outputs x if $T \subseteq [n]$ of size at least t.
- For any x and x' and a set $S \subset [n]$ of size at most $t-1$, the two distributions

$$\{(s_1, ..., s_n) \leftarrow \text{Share}(x) : (s_i | i \in S)\} \equiv \{(s_1, ..., s_n) \leftarrow \text{Share}(x') : (s_i | i \in S)\}$$

11.3.2 Shamir's Secret Sharing Protocol

Shamir's secret sharing protocol is a (t, n) secret sharing scheme given by two PPTs. The Share(s) is executed as follows. Let $\mathbb{F}$ be a finite field of size $> n$. Let $\alpha_1, \ldots, \alpha_n$ be distinct non -zero elements from $\mathbb{F}$. Share samples $t-1$ elements $a_1, a_2, ..., a_{t-1} \leftarrow \mathbb{F}$. It then defines a polynomial

$$p(x) = s + a_1 x + a_2 x^2 + ... + a_{t-1} x^{t-1}.$$

Share(x) outputs $p(\alpha_1), \ldots, p(\alpha_n)$.

The reconstruction function corresponds to polynomial interpolation since any degree $t-1$ polynomial is uniquely determined from any set of t evaluation points. The security follows since for any subset of $t-1$ points, the evaluations of p are just random elements from the finite field.

11.4 Information-Theoretic Multiparty Secure Computation

Previously we used OT which required cryptographic assumptions. In this section we will use Shamir's secret sharing protocol to show a method for multi-party secure computation without using any cryptographic assumptions. However, the catch will be that the number of parties that we allow the adversary to corrupt is at most $n/2$.

Let t be the number of adversarial corruptions and let $n = 2t+1$ be the total number of parties in the protocol. Let $x_i \in \mathbb{F}$ be the private input of the corrupt party. Similar to before, parties wish to determine $f(x_1, x_2, ..., x_n)$ without revealing their private inputs. We will use a circuit to compute f with $+$ and $*$ gates.

11.4.1 Input Sharing Phase

Each party P_i computes a $(t+1)$-out-of-n secret sharing of its input.

11.4.2 Maintaining Wire Values and Invariant

Similar to 11.2.2, parties maintain wire values for each corresponding operation result. The invariant in this case is given as follows. The output wire value of a computation given on a wire w is individually computed as w_i by party P_i and has the following property. Let the polynomial of degree t passing through the points

$$(\alpha_1, w_1), (\alpha_2, w_2), ..., (\alpha_n, w_n)$$

be denoted by q. Then the wire value $w = q(0)$. Further, no set of t parties learn any information about w.

11.4.3 Computing the Function Value

We will show that parties determine their wire values using a recursive method. As there are two operations, we split this into two cases. One where the operation is addition and the other where the operation is multiplication.

Initial Conditions For a input wire w for an arbitrary party P_i, the wire values are given by $w_j = q_i(\alpha_j)$. From the way we constructed these values, observe that the invariant is preserved.

Case 1: The operation is addition.
Suppose the operation takes input two values $p(0)$ and $q(0)$. Then the result of the operation is $p(0)+q(0)$ and parties compute the wire values as follows

$$w_i = p_i + q_i$$

A party P_i can do the operation $p_i + q_i$ locally to compute the value of the wire corresponding to $p(0)+q(0)$. One can see that the function passing through the points (α_i, p_i+q_i) passes through the point $(0, p(0)+q(0))$ thus the invariant is preserved.

Case 2: The operation is multiplication.
Now let's suppose that the operation in question is multiplication. Thus the result of the operation is $p(0)q(0)$. The parties first compute

$$w_i' = p_i q_i$$

Observe that the polynomial pq is of degree $2t$ and hence, the parties hold a $(2t+1)$-out-of-n secret sharing. However, we want the parties to hold a $(t+1)$-out-of-n secret sharing of the wire values. To solve this problem, we use a degree reduction technique.

Degree Reduction. Let's say that the parties have a $(2t+1)$-out-of-n and $(t+1)$-out-of-n secret sharing of the same random value r. Let us denote them by $(r_1', \ldots, r_n')$ and $(r_1, \ldots, r_n)$ respectively. The parties compute $w_i' - r_i'$ and exchange this with other parties. The parties reconstruct this to obtain $p(0)q(0) - r$. This is random and provides no information about $p(0)q(0)$ as r is random. The parties now compute $p(0)q(0) - r - r_i$. This is a $(t+1)$-out-of-n secret sharing of $p(0)q(0)$.

The parties can generate a $(2t+1)$-out-of-n and $(t+1)$-out-of-n secret sharing of a random value and then add these shares together to generate $(r_1', \ldots, r_n')$ and $(r_1, \ldots, r_n)$.

11.4.4 Reconstructing the Output

The parties exchange their output wire value with all the other parties. Then each party inputs these values into the Recon PPT to determine the function value.