Secure Multi-Party Computation

Notes by Yael Kalai

MIT - 6.5610
Lecture 12 (March 12, 2025)

Warning: This document is a rough draft, so it may contain bugs. Please feel free to email me with corrections.

Outline

- Motivation
- Definition
- Construction (due to Ben-Or, Goldwasser and Wigderson (BGW))

Motivation

Suppose a set of n parties, denoted by $P_1, \ldots, P_n$, with private inputs $x_1, \ldots, x_n$ want to compute a function $f(x_1, \ldots, x_n)$ without revealing any information about their secret inputs beyond the function output $f(x_1, \ldots, x_n)$. This task can be achieved using a multi-party computation (MPC) scheme. MPC enables multiple parties to jointly compute a function over their inputs while keeping their inputs private. It has several practical applications across various domains. Some examples are:

- Privacy-Preserving Data Analysis: MPC allows multiple parties to analyze sensitive data collaboratively without revealing individual inputs. This is particularly useful in sectors like healthcare, finance, and market research, where data privacy is critical but collaborative analysis is necessary.
- Secure Machine Learning: MPC can be used to train machine learning models on combined datasets from multiple sources without sharing the raw data. This is beneficial in situations where data sharing is restricted due to privacy concerns or regulations.
- Secure Auctions and Bidding: In auction scenarios, MPC ensures fairness and privacy by allowing bidders to submit encrypted bids, preventing bid manipulation and preserving bidder anonymity.

Definition

We need to define the model of communication and security.

The Model. In the MPC model we assume that all n parties are connect via *private and authenticated point-to-point channels*, which means that each pair of parties P_i and P_j can send each other messages via their point-to-point channel in a way that only they can send and receive the bits that are communicated, and no other party can read or tamper with these messages. The reason we assume the existence of such channels is that we know how to implement them using encryption schemes and signature schemes, the former to make the channel private and the latter to make it authentic.

We also assume a synchronous network, where the protocols proceed in "rounds", and at the end of each round all the messages that were sent were received. This assumption is not necessarily realistic and is made to simplify protocol design. There are various techniques in the literature for dealing with the asynchronous setting (which we will not get into in this class).

Defining Security. As we saw in the course of the semester, defining security of cryptographic primitives is tricky. The case of MPC is no different and in some sense is the trickiest of them all.

Take 1: A MPC protocol is secure if by the end of the protocol no party learns any information about the other parties' inputs.

Problem: This is impossible to achieve since the output itself $f(x_1, \ldots, x_n)$ reveals information about the other parties' inputs.

Take 2: A MPC protocol is secure if by the end of the protocol no party learns any information about the other parties' inputs, except of $f(x_1, \ldots, x_n)$.

Problem: This does not provide the desired security guarantee. We want the inputs to be "independent." Consider for example the auction setting, and suppose we do an MPC to compute who is the party with the highest bid. It is not enough to say that I don't learn anyone's bid. We need to ensure that I cannot somehow outbid you by 1 (without knowing your bid).

Take 3: A MPC protocol should be as secure as an *"ideal world"* implementation, where each party P_i sends her secret input x_i to a trusted party and then the parties receive $f(x_1, \dots, x_n)$. This is formalized below

Definition 1. An n-party protocol securely computes a function f in the presence of at most t corruptions, if for every (PPT) adversary $\mathcal{A}$ "controlling" at most t parties in the real world, there exists a (PPT) simulator $\mathcal{S}$ "controlling" the same subset of parties in the ideal world, such that for any set of inputs $x_1, \dots, x_n$,

$$\mathsf{Real}_{\mathcal{A}}(x_1, \dots, x_n) \equiv \mathsf{Ideal}_{\mathcal{S}}(\mathsf{x}_1, \dots, \mathsf{x}_\mathsf{n})$$

where $\mathsf{Real}_{\mathcal{A}}(x_1, \dots, x_n)$ is the output of all the parties after running the protocol, where the adversarial party output whatever they want; $\mathsf{Ideal}_{\mathcal{S}}(x_1, \dots, x_n)$ is the output of all the parties after handing their inputs to the trusted party, who computes f on their input and returns the output y. The honest parties output y and the adversarial parties (controlled by the simulator) output whatever they want.

As briefly mentioned in the previous lecture, we consider two types of adversaries: *honest-but-curious* and *malicious*. A malicious adversary that controls t-parties completely controls them and can choose to send arbitrary messages on their behalf (and so can the simulator in the ideal world). In the honest-but-curious setting an adversary that controls t parties learns all their inputs and the messages that they receive and send (as well as their internal state), but cannot modify their messages.

In this class we present the MPC protocol due to Ben-Or, Goldwasser and Wigderson **?** (henceforth referred to as the BGW protocol). They constructed a version that is secure against an honest-but-curious adversary that controls less than $t = n/2$ parties. They also constructed a protocol that is secure against a malicious adversary that controls less than $t = n/3$ parties. We will only cover their protocol in the honest-but-curious setting.

We note that their protocol is information theoretically secure! It does not rely on any cryptography, and is secure even if the adversary is all powerful.

We note that if we rely on cryptography we can get security against any number of corruptions!

The BGW Protocol

The high-level idea behind the BGW is the following. It consists of two stages:

Phase 1: Secret-sharing of the inputs. First, each party secret-shares her input among the n parties. They use the $t+1$-out-of-n secret sharing scheme of Shamir (that we learned in class). Recall that an adversary that controls at most t parties learns nothing (information theoretically).

Formally, each party P_i does the following:

1. Select uniformly t coefficients $a_1, \dots, a_t$ from $\mathsf{GF}[p]$, and let $q_i(x) = s_i + a_1 x \cdots + a_t x^t$.
2. Send $q_i(j)$ to party P_j, for all $j \in [n]$.

Phase 2: Computation on shares. Next the parties jointly compute the function f. Think of f as being represented as an arithmetic circuit with addition gates and multiplication gates modulo 2. The parties will jointly compute each gate in a way, that given shares of the input wires to the gate they end up with shares of the output wire to the gate.

Addition gates: Let g be an addition gate with input gates with values $p(0)$ and $q(0)$. Assume parties hold shares of $p(0)$ and $q(0)$ (i.e., party P_i holds $p(i)$ and $q(i)$). Each party P_i computes a secret share of $(p+q)(x)$ (which is a secret share of $p(0) + q(0)$), by setting the new share to be $p(i) + q(i)$. Note that no interaction was required here! The parties did this computation locally.

Remark. Note that the addition is modulo p and not modulo 2. But that is ok since for every $a, b \in \{0, 1\}$,

$$a + b \ (mod\ 2) = a + b - ab \ (mod\ p)$$

We also note that the parties can compute a gate that is multiplication by scalar (modulo p) by simply multiplying each share (locally) by that scalar.

Multiplication gates: Let g be a multiplication gate with input gates with values $p(0)$ and $q(0)$. Assume the parties hold shares of $p(0)$ and $q(0)$ (i.e., party P_i holds $p(i)$ and $q(i)$). It is tempting to simply multiply the shares and set the share of gate g be $p(i) \cdot q(i)$. The problem is that this increases the degree of the polynomial from degree t to degree $2t$. So, now we will need $2t + 1$ parties to reconstruct, and this number would grow with every multiplication gate. Nevertheless, this is the first step: Each party P_i computes $v_i = p(i) \cdot q(i)$. Then, the parties reduce the degree of the secret sharing polynomial from $2t$ to t. This step involves interaction. Actually, all the parties need to do is to *share the shares*! Each party P_i secret shares v_i using a t-out-of-n Shamir secret sharing scheme. Next class we will argue that this is enough to locally compute shares of of the output gate with respect to a degree t polynomial.