CS 7880 Graduate Cryptography September 13, 2017

Lecture 3: Multiparty Computation

Lecturer: Daniel Wichs *Scribe: Eysa Lee*

1 Topic Covered

- Definition of Multi-Party Computation (MPC)
- Building a MPC protocol using Shamir secret sharing
- Definition of statistical distance

2 Multi-Party Computation (MPC)

Let us begin by considering a scenario where a group of students is trying to decide whether or not to ask their teacher to extend a deadline on an assignment. The students decide that they should only approach their teacher if the majority of them have not yet finished, but no one wants to openly reveal whether or not they have completed the assignment. The students wish to determine where the majority lies without revealing anyone's completion status or even how many are in each category. It turns out we can achieve this using a **multi-party computation (MPC)** protocol.

We generalize our example as n parties that want to evaluate some function f on secret inputs $x_1 \dots x_n$. In computing $f(x_1, \dots, x_n)$, the parties want to do so without revealing any additional information, including each of their secret inputs to each other. A MPC protocol π for f has the following properties:

- **Correctness** After running π, all parties learn $f(x_1, \dots, x_n)$.
- **Perfect, honest-but-curious, t-collusion security (t-security)** Let $[n]$ denote the set $\{1, \dots, n\}$, and t be the number of adversarial parties colluding.

  For all sets $S \subseteq [n]$, where $|S| = t$, there exists a simulator Sim such that for all $x_1, \dots, x_n$:
  $$\mathsf{View}_S(\pi(x_1, \dots, x_n)) \equiv \mathsf{Sim}(x_S, f(x_1, \dots, x_n)).$$
  The random variable $\mathsf{View}_S(\pi(x_1, \dots, x_n))$ denotes the view of the parties $i \in S$ during the execution of the protocol and in particular contains their inputs and randomness as well as all of the messages they see during the execution of the protocol. The $\equiv$ sign denotes distributional equivalence.

For t-security, the intuition is that an adversary should not learn more from participating in the protocol than it could have gotten from the secret inputs of the corrupted parties x_S and the final output $f(x_1, \dots, x_n)$. This intuition is formalized by saying that the distributions of the view of the adversary on the protocol should be equivalent to what could be produced by a simulator given only x_S and $f(x_1, \ldots, x_n)$.

We note that for t-security, we consider the *honest but curious* model. In this model, the adversarial parties are trusted to run the protocol as specified but may be 'curious' (i.e. want to learn information) about the other parties. We let this model be the default for t-security in this lecture but later in the course we will also consider a stronger model where adversarial parties can deviate arbitrarily from the protocol execution.

We now show that, when we have an honest majority $t < n/2$, then for all functions f there exists a MPC protocol for f.

Theorem 1 *$\forall t$, n such that $t < n/2$, $\forall f$, $\exists$ a t-secure MPC π for f.*

We show this by constructing a MPC protocol π for any arbitrary function f using Shamir secret sharing in the next section.

2.1 Construction

Recall that with Shamir secret sharing, random coefficients were generated to define a unique polynomial of degree t, and Lagrange interpolation enabled recovery of the polynomial given $t+1$ points. To assist in our construction, we choose to represent f as an arithmetic circuit that contains either addition or multiplication gates. Any circuit can be represented in this manner and any polynomial-time computation can be represented by a polynomial size arithmetic circuit.

Each input wire is associated with a party $i \in [n]$ but it may be the case that one party's input may consist of several field elements that go on different input wires.

We construct π as follows.

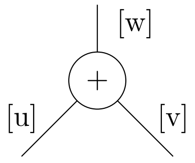
1. Each party begins by using t-threshold Shamir secret sharing to share each[1] of its input values. That is, for each input u, it creates a secret sharing of u as

$$[u] = (p(1), \ldots, p(n))$$

   where p is a random polynomial of degree $\leq t$ and $p(0) = u$. It sends the share $u_i = p(i)$ to party i.

2. Once each party has a secret share of each of the inputs, the parties proceed to compute secret shares for the values on each internal wire of the circuit. For some gate within the circuit assume the parties already computed secret sharing of the inputs to that gate: $[u] = (u_1, \ldots, u_n)$ and $[v] = (v_1, \ldots, v_n)$. Their goal is to compute a secret sharing $[w] = (w_1, \ldots, w_n)$ of the output of the gate. We consider each of these three cases:

   (a) Addition



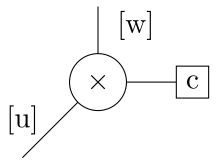
[1]"Each" in the sense that we do not restrict to a single field element

In this case we can set $[w] := [u]+[v] = (u_1+v_1, \dots, u_n+v_n)$. In other words each party individually adds up its shares of u and v to get a corresponding share of w. Note that $[u] = (p(1) \dots p(n))$ and $[v] = (q(1) \dots q(n))$ for some polynomials p, q of degree $\leq t$ such that $p(0) = u$ and $q(0) = v$. Therefore

$$[w] = ((p+q)(1) \dots (p+q)(n))$$

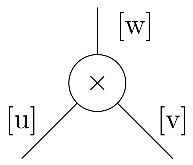
where $(p+q)$ is a degree $\leq t$ polynomial such that $(p+q)(0) = u+v = w$ as desired.

(b) Multiplication by a constant



In this case we can set $[w] := c \cdot [u] = (c \cdot u_1, \dots c \cdot u_n)$. Each party individually multiplies its share of u by the constant c.

(c) Multiplication



For multiplication, things become more tricky. The parties can compute:

$$[u] \times [v] = (u_1 v_1, \dots, u_n v_n) = (p(1)q(1) \dots p(n)q(n)) = ((pq)(1) \dots (pq)(n))$$

by multiplying their individual shares of u, v. Note that the polynomial pq does satisfy $(pq)(0) = u \cdot v = w$ as desired but it is of degree $2t$ rather than t. Since we chose $n > 2t$ the n shares are sufficient to uniquely recover w. But we cannot simply set $[w] = [u] \times [v]$ and keep going with the computation since the next multiplication would then increase the degree to $4t$ which is too large and we would no longer be able to recover the correct value from the shares. Instead, we use a neat trick to reduce the degree of the polynomial.

The parties first compute $[z] = (z_1 \dots z_n) = [u] \times [v]$. Then each party creates a fresh t-out-of-n Shamir secret sharing $[z_i] = (z_{i,1} \dots z_{i,n})$ of its own share z_i and sends the share $z_{i,j}$ to party j. In other words it chooses a random polynomial p_i of degree $\leq t$ such that $p_i(0) = z_i$ and sets $z_{i,j} = p_i(j)$. We can then set define the secret shares of w to be $[w] = \sum \alpha_i [z_i] = (\sum \alpha_i z_{i,1}, \dots, \sum \alpha_i z_{i,n})$, where α_i is a constant determines by the Lagrange interpolation relation $w = \sum \alpha_i z_i$. Note that $[w] = (p^*(1)), \dots, p^*(n))$ where $p^* = \sum_i \alpha_i p_i$ is a degree $\leq t$ polynomial such that $p^*(0) = \sum \alpha_i p_i(0) = w$.

3. Finally, each party broadcasts their secret share of each output wire $[y]$ which allows all the parties to recover the outputs.

We have just constructed a protocol to compute an arbitrary f represented as an arithmetic circuit. We claim this is secure.

Theorem 2 *The above protocol is t-secure.*

Proof: Recall the security definition of MPC:

$$\mathsf{View}_S(\pi(x_1, \ldots, x_n)) \equiv \mathsf{Sim}(x_s, f(x_1, \ldots, x_n))$$

Note that $\mathsf{View}_S(\pi(x_1, \ldots, x_n))$ consists of values $x_S, r_S, [u]_S, [z_i]_S, [y]$, where x_S are the inputs of the adversarial parties, r_S is randomness of the adversarial parties, for each input of an honest party $[u]_S$ denotes that shares seend by the adverarial parties, $[z_i]_S$ are the shares of shares seen by the adversarial parties during the computation of the each multiplication gate, and $[y]$ are the shares for each output wire of all n parties.

The simulator is given x_S, so simulating the input of the colluding parties from the view is trivial: the simulator simply uses the given x_S. We note that the distributions of secret shares $[u]_S$ and $[z_i]_S$ are uniformly random by the t-out-of-n security of Shamir secret sharing, so r_S, $[u]_S$, and $[z_i]_S$ can be simulated by choosing values at random.

For $[y]$, we can not choose this value uniformly randomly since it is dependent on other values in the protocol. However, the simulator can compute the shares $[y]_S$ of the parties in S from our other sampled values by executing the protocol on their behalf. Recalling $|S| = t$, computing $[y]_S$ gives us t points of the polynomial. The simulator is given the output $y = f(x_1, \ldots, x_n)$, which gives us an additional point on the polynomial. With these $t + 1$ points, we can compute the polynomial using Lagrange interpolation, and compute $[y]_{\bar{S}}$, which are the remaining secret shares of $[y]$. Therefore, since we are able to build a simulator that produces a distribution equivalent to an adversary's view of the protocol, the protocol is t-secure. $\square$

2.2 Impossibility: 2 Party Computation for AND

We now show that we cannot achieve perfect security in general when $t > n/2$. Let us consider an AND gate. With 1 bit inputs, we can see that AND is equivalent to a multiplication gate: the output is 1 if both input wires are 1, and 0 otherwise.

Theorem 3 *For the* AND *function with $n = 2$ parties, there is no perfectly t-secure MPC protocol.*

Proof: Consider two parties Alice and Bob with inputs x_a and x_b, respectively. They communicate back and forth to compute $x_a \wedge x_b$. Let T be the transcript of the protocol. We say that T is consistent with (e.g.,) $x_a = 0$ if there is some randomness that Alice could have used had she had the input 0 that would have made her send the responses in T.

If Alice's bit $x_a = 1$, then T can only be consistent with one of $x_b = 0$ and $x_b = 1$. This follows by correctness. Since Alice needs to output a different bit depending on whether $x_b = 0$ or $x_b = 1$ the transcript T cannot be consistent with both options. This itself is not a contradiction.

If Alice's bit $x_a = 0$, then T has to be consistent with both $x_b = 0$ and $x_b = 1$. This follows by security since Alice should not learn Bob's bit and if the transcript was only consistent with one of the two options a computationally unbounded Alice could figure out which one.

But the above two conditions imply that Bob can learn Alice's bit when $x_b = 0$. In particular, at the end of the protocol Bob checks whether T is consistent with only one of $x_b = 0, x_b = 1$ or both of them. By the above this completely reveals Alice's bit which Bob should not have learned in a secure MPC when his input is $x_b = 0$. $\square$

Note that the above argument does not lead to an *efficient* attack and hence this only precludes perfectly secure MPC protocols. Later on in the course we will see how to get security against computationally bounded adversaries.

We can extend this impossibility result to showing that for any n there is no perfectly secure MPC protocol for the n-party AND function defined as $f(x_1, \ldots, x_n) = \bigwedge_{i=1}^{n} x_i$ when the collusion size is $t \geq n/2$. How? If there was one we could convert it into a 2-party protocol for the AND function which is secure against 1 adversarial party. We can let Alice act as the first $n/2$ parties in the protocol each of which uses Alice's input $x_1 = x2 = \ldots x_{n/2} = x_a$ and Bob act as the remaining $n/2$ parties each of which uses Bob's input $x_{n/2+1} = \cdots = x_n = x_b$. Note that an adversarial Alice or Bob sees the view of $n/2$ parties in the n-party protocol but this preserves security if the n-party protocol is secure against $n/2$-size collusions. Since we already saw that it is not possible to get 2-party MPC for the AND function, it is not possible to get an n-party MPC for the AND function with collusion size $t \geq n/2$.

3 Statistical Distance

Recall the definition of perfect security:

DEFINITION 1 An encryption scheme has perfect security if for all m_0, m_1,

$$\mathsf{Enc}(K, m_0) \equiv \mathsf{Enc}(K, m_1)$$

$\diamondsuit$

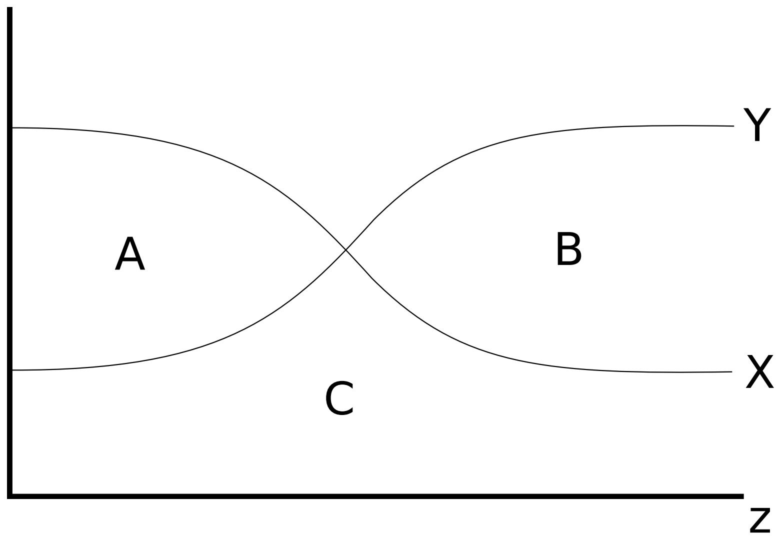
Although perfect security is great if we can get it, we already saw that it requires the key to be at least as large as the message. We will begin looking at ways to relax the requirements of perfect security. The first attempt at relaxing the requirement relies on the notion of statistical distance.

DEFINITION 2 For random variables X, Y with support $\mathcal{Z}$ the statistical distance SD of X, Y is:

$$\mathsf{SD}(X, Y) = \frac{1}{2} \sum_{z \in \mathcal{Z}} \Big| \Pr[X = z] - \Pr[Y = z] \Big|.$$

$\diamondsuit$

Let us draw a sample of what X, Y might look like if we "sort" the support by the difference between the probability assigned to each point by X vs Y.



Letting $|A|$ be the area of A, we observe that $|A| + |C| = 1$, $|B| + |C| = 1$, and therefore $|A| = |B|$. This shows that

$$\mathsf{SD}(X,Y) = \frac{1}{2}\Big|\big|A\big| + \big|B\big|\Big| = |A| = |B|$$

To get inuition for this definition, imagine we were trying to distinguish between X and Y. That is we wanted to find the set D that maximizes $\Big|\Pr[X \in D] - \Pr[Y \in D]\Big|$. Then this is achieved by setting $D = A$ or $D = B$ (clear from picture). Therefore we get the equivalent definition

$$\mathsf{SD}(X,Y) = \max_{D \subseteq \mathcal{Z}}\Big|\Pr[X \in D] - \Pr[Y \in D]\Big|.$$

Alternately, we can think of a distinguisher D as an arbitrary function that outputs either 0 or 1. We want to maximize the difference in probabilities between outputting 1 on input X vs Y. Since we can equate any such function with the set of points on which it outputs 1 this gives the equivalent definition

$$\mathsf{SD}(X,Y) = \max_{D\,:\,\mathcal{Z} \to \{0,1\}}\Big|\Pr[D(X) = 1] - \Pr[D(Y) = 1]\Big|.$$