Secure Multi-Party Computation (Part I)

1 Formalizing Cryptographic Protocol Problems

Cryptographic protocols attempt to solve problems where two or more mutually distrustful parties wish to carry out some process (e.g., compute a function) in a secure way: that is, with the guarantee that the process is carried out fairly and without revealing any private information.

A general way of specifying a cryptographic protocol for k parties is as a probabilistic process with k inputs and k outputs.

1.1 Functionality

Definition 1 (k-Party Functionality) *A k-party functionality is a functionality*

$$f : (\{0,1\}^*)^k \to (\{0,1\}^*)^k$$

that is computable in probabilistic polynomial time.

The k-party functionality takes as its input the private information of each of the k parties and produces as output a vector of the k values required by the parties to solve the problem at hand.

For convenience, we will sometimes use the notation $f_i(\cdot)$ to indicate the ith element of f's output vector—the information intended for party i. Additionally, some functionalities may not depend on inputs from or provide outputs to particular parties, and to make these situations explicit, we will denote the corresponding inputs or outputs as 0 rather than as some variables or expressions.

Example 1 (Coin-Flipping) In coin-flipping, the two parties wish to agree on a random bit, but they want to be sure that neither can unfairly bias this bit. The formulation as a functionality is $f(0,0) = (r,r)$ where r is random and uniform in $\{0,1\}$.

Example 2 (Election) In an election, each of the k parties has a "vote" that is a bit. The desired outcome is for everyone to know the sum of all of the parties' votes but nothing more (e.g., any other party's vote). As a functionality, if $v_j \in B$ is the vote of party j, then, for all $1 \le i \le k$, $f_i(v_1, v_2, \ldots, v_k) = \sum_{j=1}^{k} v_j$.

1.1.1 The Obvious Problem

If the parties trusted each other, they could just share all of the inputs and compute the functionality themselves. However, since it is assumed that the parties do not all trust each other, some mechanism must be employed in the computation of this functionality to ensure that it is evaluated fairly and without spilling anyone's secrets.

In the sequel, we explore how this might be accomplished.

1.2 The Ideal Model

Ideally, the parties could take their problem to a trusted third party T. After they gave T all of their private information, she would compute the functionality and distribute the outputs to just the appropriate parties.

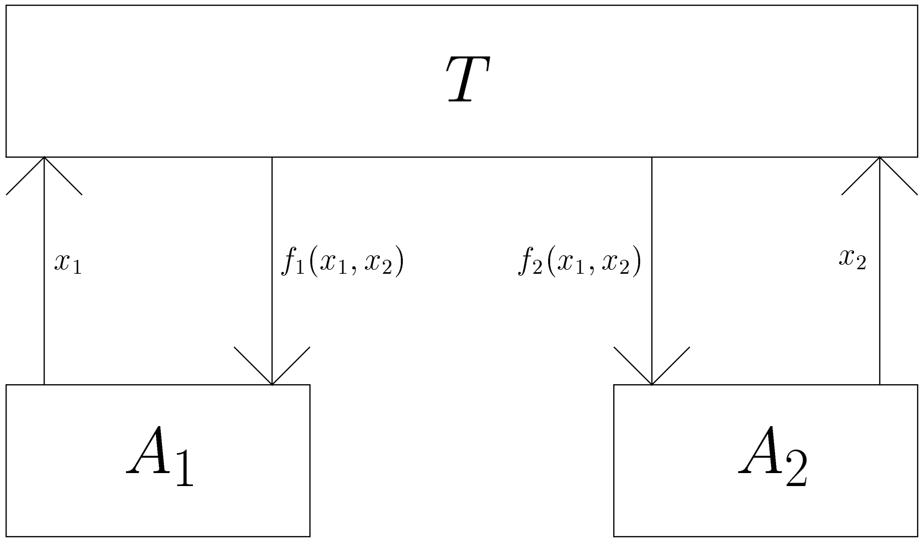

Figure 1: The Ideal Model: T computes f privately for A_1 and A_2

1.3 The Real World

Cryptographic protocols would be simple if there was an ever present, always willing, universally trusted, third party. Since this is not the case, we need different models in order to discuss reality.

We consider two "real world" models:

Semi-Honest Model In the semi-honest model, participants follow the protocol exactly, but they may remember all intermediate messages and computations and use these in an attempt to learn more than they would know after computing the functionality in the ideal model.

Malicious Model The malicious model, unlike the semi-honest model, permits parties up to $k - 1$ of the parties to deviate from the protocol in arbitrary ways: they may lie about their inputs or simply refuse to participate at any point in the protocol. A protocol which is secure in the malicious model will not yield any information to the deviant participants that they would not get in the ideal model.

In this lecture, we will consider only the semi-honest model. Although it at first seems to have limited applicability in the real world, we shall later see that protocols that are secure in the semi-honest model can be used to construct protocols that are secure even in the presence of malicious parties.

1.3.1 Definition of Security in the Semi-Honest Model

Intuitively, a protocol is secure in the semi-honest model if a group of parties which follow the protocol are unable to learn anything that they could not in the ideal model. In an attempt to capture this notion, we say that given only a particular party's input and output, it is possible *simulate* his view of the entire protocol without knowing any other party's input or output. Since he cannot distinguish his view of simulation (which does not depend on others' secrets) from his view of protocol (which does, in general, depend on the other parties' secrets), we claim he does not learn anything beyond what he should by participating in the protocol.

To see formally what it means for a protocol to be secure in the semi-honest model, we will first consider the case of a two-party protocol.

Let $f : (\{0,1\}^*)^2 \rightarrow (\{0,1\}^*)^2$ be a 2-party functional.

Definition 2 (Protocol) *A protocol for f is a pair of probabilistic polynomial time algorithms A_1 and A_2 such that for all x_1 and x_2, if we let $A_1(x_1)$ interact with $A_2(x_2)$, $A_1(x_1)$ outputs $f_1(x_1, x_2)$ and $A_2(x_2)$ outputs $f_2(x_1, x_2)$.*

The condition in the above definition simply says that the protocol is *correct* in computing the functionality f. In discussing secure multi-party computation, it is common to take this property for granted and focus instead on the protocol's *security*.

Definition 3 (Security in the Semi-Honest Model (Two Parties)) *A protocol A_1, A_2 for f is* secure *is there are two polynomial time probabilistic* simulators *S_1 and S_2 such that, for all x_1 and x_2, the view of A_1 in the interaction of $A_1(x_1)$ with $A_2(x_2)$ is computationally indistinguishable from $S_1(x_1, f_1(x_1, x_2))$. Similarly, the view of A_2 in the interaction of $A_1(x_1)$ with $A_2(x_2)$ is computationally indistinguishable from $S_2(x_2, f_2(x_1, x_2))$.*

The definition of security for k parties is similar. Rather than having just k simulators, however, we will need $2^k - 2$, one for each nontrivial subset of the parties: for any interesting colluding group of the parties, there is a simulator that, given only their inputs and outputs, can provide a view that is indistinguishable from their view in an actual run of the protocol.

Let $f : (\{0,1\}^*)^k \rightarrow (\{0,1\}^*)^k$ be a k-party functional.

Definition 4 (Security in the Semi-Honest Model (k Parties)) *A protocol $A_1, \ldots, A_k$ for f is* secure *if for all $I \subset \{1, \ldots, k\}$ with $I \neq \emptyset$, there is a probabilistic polynomial time simulator S_I such that for all $x_1, \ldots, x_k$, $S_I(x_I, f_I(x_1, \ldots, x_k))$ is computationally indistinguishable from the joint views of $\{A_i(x_i) \mid \forall i \in I\}$ in their interaction with the other $k - |I|$ parties in the protocol.*

This definition of k-party security says that an adversary who views some subset of the k parties in the protocol does not learn anything from their participation in the protocol that he would not learn in the ideal model.

2 Realization of the Ideal

Our ultimate goal is to devise some procedure for producing a cryptographic protocol which securely computes any k-party functionality in the malicious model.

The goal is reached through a series steps. Working with intermediate models like the semi-honest model allow us to solve particular parts of the problem before addressing others.

In this lecture, we take a first step: we show that any 2-party functionality can be computed securely in the semi-honest model.

The result relies on the assumption of the existence of trap-door one-way permutations.

2.1 Oblivious Transfer

First we need to introduce the primary cryptographic tool used in constructing 2-party protocols which are secure in the semi-honest model.

Definition 5 ((k-to-1) Oblivious Transfer) *The task of* (k-to-1) oblivious transfer *is computing the 2-party functionality*

$$f : \{0,1\}^k \times \{1,\ldots,k\} \to \emptyset \times \{0,1\}$$

where $f((b_1,\ldots,b_k),i) = (0,b_i)$.

That is, A has k bits $b_1,\ldots b_k$, and B wants to learn bit i. In oblivious transfer, B learns only b_i and A does not know which bit B learned.

Theorem 1
If one-way trapdoor permutations exist then (k-to-1) Oblivious transfer has a 2-party protocol that is secure in the semi-honest model.
Proof: Let f be a trapdoor permutation with key generator G and inverter I.

A gets input $b_1,b_2,\ldots,b_k \in \{0,1\}$. B gets input $i \in \{1,\ldots,k\}$. The protocol proceeds as follows:

1. A generates the keys $(p_k,s_k) \leftarrow G$ and sends p_k to B
2. B picks $r_1,\ldots,r_k$ and $z_1,\ldots,z_k$ all independently and uniformly at random from $\{0,1\}^n$. B sends $(r_1,\ldots,r_k)$ and $(z_1,\ldots,F(p_k,z_i),\ldots,z_k)$ to A.
3. A sends $m = (b_1 \oplus (r_1 \cdot I(s_k,z_1)),\ldots,b_i \oplus (r_i \cdot I(s_k,F(p_k,z_i))),\ldots,b_k \oplus (r_k \cdot I(s_k,z_k)))$ to B.
4. B outputs $m_i \oplus (r_i \cdot z_i)$

At the end of the protocol, B's output is correct since

$$\begin{aligned} m_i \oplus (r_i \cdot z_i) &= b_i \oplus (r_i \cdot I(s_k,F(p_k,z_i))) \oplus (r_i \cdot z_i) \\ &= b_i \oplus (r_i \cdot z_i) \oplus (r_i \cdot z_i) \\ &= b_i. \end{aligned}$$

To see why this is secure, note that since F is a permutation, A cannot tell which coordinate in B's message in step 2 is the output of F, thus B's desired index i remains a secret.

For all of the bits besides b_i, A has essentially only sent Goldwasser-Micali encryptions of them to B, so B cannot recover them much more easily than he can invert F without s_k. $\square$

Note that the above protocol is extremely vulnerable to malicious parties.

2.2 Secure 2-Party Semi-Honest Computation

Now the primary result of this lecture:

THEOREM 2
If trapdoor one-way permutations exist, any 2-party functionality can be computed securely in the 2-party semi-honest model.

PROOF:
Let $f : (\{0,1\}^*)^2 \to (\{0,1\}^*)^2$ be an arbitrary 2-party functionality.

Since f has a probabilistic algorithm that runs in polynomial time, we can also think of it as being computed by a polynomial-sized circuit C composed of AND-gates and XOR-gates, each with fan-in 2; and NOT-gates with fan-in 1.

The two algorithms which will carry out the protocols will be called A and B. A has inputs $\{x_i : 1 \le i \le n\}$ and B has inputs $\{x_i : n+1 \le i \le 2n\}$.

We consider the values in the computation as integers modulo 2, so the gates can be defined:

- $\text{AND}(x, y) \equiv x \cdot y \pmod 2$
- $\text{XOR}(x, y) \equiv x + y \pmod 2$
- $\text{NOT}(x) \equiv x + 1 \pmod 2$

The parties need to distribute their inputs to each other in a secure way. In the first step of the protocol, A picks n random bits $r_1, \ldots r_n$, and sends them to B, who calls them $b_1, \ldots b_n$. For $1 \le i \le n$, A sets $a_i = r_i + x_i$. At this point, for $1 \le i \le n$, $x_i = a_i + b_i$.

Similarly, B picks n random bits $r_{n+1}, \ldots, r_{2n}$, and sends them to A, who calls them $a_{n+1}, \ldots, a_{2n}$. For $(n+1) \le i \le 2n$, B sets $b_i = x_i + r_i$. Now $x_i = a_i + b_i$ for $1 \le i \le 2n$; the parties have distributed the inputs. Note that A sees only random bits $(a_{n+1}, \ldots a_n)$ for B's half of the input and B sees only random bits for A's half.

The protocol will proceed by computing the value of one wire in the circuit each round. A and B share the values on the wires just like they share the inputs: the true value of a particular wire will be the sum modulo 2 of the values computed by A and B.

At this point it is sufficient to show that a shared value of the output of each of the three types of gates can be computed given the shared inputs without disclosing the true value of any bit. As we shall see, the NOT and the XOR gates are easy to compute since they are linear functions, while the AND gate is trickier.

In the following, we assume that x_1 and x_2 are inputs to the gates, that A's shares are a_1 and a_2, and that B has $b_1 = x_1 + a_1$ and $b_2 = x_2 + a_2$, so that $x_1 = a_1 + b_1$ and $x_2 = a_2 + b_2$ (recall that all arithmetic is modulo 2). The goal is for A and B to compute a_3 and b_3 respectively, so that the the value of the output of the gate is $a_3 + b_3$. Neither side should learn anything about the other's inputs or the true value of the output.

NOT To compute $x_3 = \text{NOT}(x_1)$, it is sufficient for just one side to complement its bit: we can assume A does this, so that $a_3 \leftarrow (a_1 + 1)$, while B simply assigns $b_3 \leftarrow b_1$. No communication is necessary, so nothing is learned by either party. The shared value is correct since:

$$\begin{aligned} a_3 + b_3 &= (a_1 + 1) + b_1 \\ &= (a_1 + b_1) + 1 \\ &= x_1 + 1 \\ &= \mathrm{NOT}(x_1) \end{aligned}$$

XOR To compute $x_3 = \mathrm{XOR}(x_1, x_2)$, each side independently computes the XOR of its half of the input: A computes $a_3 \leftarrow (a_1 + a_2)$ while B computes $b_3 \leftarrow (b_1 + b_2)$. No communication is necessary, so nothing is learned by either party. The shared value is correct since:

$$\begin{aligned} a_3 + b_3 &= (a_1 + a_2) + (b_1 + b_2) \\ &= (a_1 + b_1) + (a_2 + b_2) \\ &= x_1 + x_2 \\ &= \mathrm{XOR}(x_1, x_2) \end{aligned}$$

AND Privately computing $x_3 = \mathrm{AND}(x_1, x_2)$ uses oblivious transfer. From A's point of view, different outputs are possible depending on what B's shares of the input wires (b_1 and b_2) are. The way to compute AND privately is for A to compute the correct (true) output of the AND for each of the possible cases of B's shares, mask them with a random bit (now A's share of the output), and use (4-1) oblivious transfer to allow B to privately request his correct share.

1. A picks $r \in \{0, 1\}$ at random, sets $a_3 \leftarrow r$, and computes

$$\begin{aligned} m_{00} &= r + a_1 \cdot a_2 \\ m_{01} &= r + a_1 \cdot (a_2 + 1) \\ m_{10} &= r + (a_1 + 1) \cdot a_2 \\ m_{11} &= r + (a_1 + 1) \cdot (a_2 + 1) \end{aligned}$$

2. Using oblivious transfer, B privately obtains $m_{b_1 b_2}$ and sets $b_3 \leftarrow m_{b_1 b_2}$.

The shared value is correct since:

$$\begin{aligned} a_3 + b_3 &= r + m_{b_1 b_2} \\ &= r + (r + (a_1 + b_1) \cdot (a_2 + b_2)) \\ &= (a_1 + b_1) \cdot (a_2 + b_2) \\ &= x_1 \cdot x_2 \\ &= \mathrm{AND}(x_1, x_2) \end{aligned}$$

It is left as an exercise for the reader to show that this computation of AND is secure in the semi-honest model (i.e., that there are simulators that mimic either side's view of the other's behavior).

□