Preliminaries and Definitions

CS 507, Topics in Cryptography: Secure Computation

David Heath

Fall 2026

Last time, we saw our first protocols that are—in some sense—secure. However, we simply waved our hands about why they are secure. Our goal today is to establish vocabulary that will allow us to capture formally the sense in which the protocols we have seen are secure.

What is a Protocol?

In this course we are concerned with the construction of algorithms, data structures, and protocols. These tools are intended to be used by parties such that they can securely execute computations of their choice. We will model each of these parties as an interactive, randomized algorithm. Often, though not always, we will consider parties that are restricted to run in polynomial time (in some parameter).

Let's describe what a protocol is. To do so, it's convenient to consider an example. Let's recall the simple two-party XOR protocol we saw last time:

Protocol 1 (Two-Party XOR)**.** *The protocol runs between two parties A and B.*

- ***Input:*** *Party A inputs a bit $x \in \{0,1\}$; B inputs a bit $y \in \{0,1\}$.*
- ***Output:*** *Both parties output $x \oplus y$.*
- ***Procedure:***
  - *A sends x to B.*
  - *B sends y to A.*
  - *A (resp. B) locally computes $x \oplus y$ and outputs the result.*

As we see by the above example, a protocol is simply an interaction between parties, where parties start from some inputs, perform local computation, communicate by exchanging messages, and eventually arrive at some output. Namely, a protocol is an interactive, randomized procedure.

A remark about communication. It's worth noting that the above protocol simply states that A sends a message to B (and vice versa). By stating it this way we have made quite strong, simplifying assumptions about how communication works. Indeed, we will, unless otherwise stated, take the following assumptions about communication:

- We assume that parties have access to secure point-to-point channels. Namely, party A knows the identity of all of its peers, can send a message to some other party B, and no other party C can read the content of that message.
- We will assume communication channels are synchronous, which simply means that a message sent from A to B will arrive at B within a fixed amount of time.

These assumptions result in protocols that are simple to design and easy to understand. Note that it is not always necessary or appropriate to take these assumptions; for example, it might be more realistic to consider communication channels that are asynchronous, where messages between parties can be arbitrarily delayed. However, for the majority of this course, we will not concern ourselves with such issues, as there is already a huge amount to discuss in our simplest setting.

What does Security Mean?

Our goal is to design cryptographic protocols that allow parties to securely compute programs. Let us temporarily assume that there are only two parties, A and B who hold respective inputs x and y. We wish to "securely" compute $F(x, y)$, where F is some program agreed upon by A and B.

When the parties A and B execute a protocol Π, they receive messages from one another. Somehow, we wish to argue that these messages are "safe to send", i.e., that somehow the messages do not violate privacy concerns. We wish to define this notion more precisely.

Our first definition of protocol security will be intended to capture the following intuitive notion:

> A protocol Π securely computes F if, by looking at messages received from B, A learns nothing about B's input y, except for that which can be learned from x and $F(x, y)$.

That is, somehow the messages A receives during Π were a consequence of her input x and the output $F(x, y)$.

Here's another way to understand the above intuitive notion. Suppose that we give to A the output $F(x, y)$, and from this A can *make up* a sequence of messages that "look the same" as the ones she sees from B during execution of Π. If that's the case, then it must be that messages from B leak nothing, except what can be learned from $F(x, y)$. Indeed, this intuitive notion of "making up" a sequence of messages will correspond closely to our eventual formal definition of security.

Before we define security more formally, let's observe that A can indeed make up messages in our example protocol:

Example 1. *In Protocol 1, A receives y from B, and she uses this to compute $x \oplus y$. But, notice that given her input x and the output $x \oplus y$, A can compute $y = x \oplus (x \oplus y)$. Thus, from Π's input/output behavior alone, A can indeed make up the message she received from B. It is in this sense that this protocol Π is secure.*

The Real World/Ideal World Paradigm

While we are not yet ready to write our formal definitions, we are ready to understand how those definitions will be structured. Typical MPC definitions operate in what we refer to as a real world/ideal world paradigm.

Recall our parties wish to compute F on their private inputs and receive only the output. Suppose our parties had access to an incorruptible, always-online, trusted third party (TTP). We will sometimes refer to such a TTP as an **ideal functionality**. Such a third party is not realistic, but if it existed, the problem of MPC would become trivial. Indeed, we could consider the following straightforward protocol that solves MPC in general:

Protocol 2 (The Ideal Protocol)**.** *The protocol takes place between parties $P_0, \ldots, P_{p-1}$ and a TTP T.*

- ***Parameters:*** *Parties agree on a function F.*
- ***Input:*** *Each party P_i has input x_i. T has no input.*
- ***Output:*** *Each party P_i outputs $F(x_0, \ldots, x_{p-1})$. T outputs nothing.*
- ***Procedure:***
  - *Each party P_i sends x_i to T.*
  - *T locally computes $y = F(x_0, \ldots, x_{p-1})$.*
  - *T sends y to each party P_i, and each party P_i outputs y.*

To re-emphasize, this protocol is not realistic, since incorruptible TTPs do not exist. However, Protocol 2 is still useful as a means by which to *define what security means*. Our formal definitions of security will compare executions of real-world protocols Π to executions of idealized protocols, such as Protocol 2.

Preliminaries

Security Parameter. As is typical in computer science, we will often analyze our protocols in an asymptotic sense. In particular, we will consider how quickly our protocols become more secure as we tune some parameter. We will refer to this parameter as the security parameter. For instance, the security parameter might dictate the length of cryptographic keys used in some underlying encryption scheme.

In this course, we will often use λ to denote a security parameter, though other variables are common in the literature, including $k, \kappa, \sigma, n, \ldots$. Note that the security parameter can be distinct from the length of logical inputs. For instance, we might have a protocol where parties can provide inputs of length n and where our security parameter is λ.

Probability Ensembles. In full generality, a protocol is a *randomized* process. Each of the parties might flip local coins and use those coins to construct messages. Hence, protocol messages are, in general, **random variables**.

Moreover, as we increase the security parameter λ, these random variables might change. This motivates the definition of an ensemble. A probability ensemble is just a collection of random variables indexed by some parameter, say λ.

Definition 1 (Ensemble)**.** *A probability ensemble (or just ensemble) X is a family of random variables X_λ where λ ranges over the natural numbers $\mathbb{N}$. That is,*

$$X = \{X_\lambda\}_{\lambda \in \mathbb{N}}.$$

In general, we will consider ensembles that are described by some process, such as details of the protocols we construct. For example, we might construct a procedure that describes all of the protocol messages that A receives in Protocol 1. In full generality, such a procedure describes an ensemble.

Now returning to the ideal world/real world paradigm, our goal will be to compare two ensembles, one describing the real world, and one describing the ideal world. Thus, we need appropriate vocabulary by which to compare ensembles as constructed by procedures. We will have three such notions:

Definition 2 (Identically Distributed)**.** *Let X and Y be two ensembles. We say that X and Y are identically distributed (or sometimes equivalent), written $X \equiv Y$, when for every λ, X_λ and Y_λ have the same distribution.*

For instance, we might have a real-world protocol where information seen by the parties is exactly the same as in the ideal-world; this will turn out to be case for Protocol 1. However, we will not always be able to achieve such a strong notion of security, and so we often need weaker notions of similarity.

To give such notions, we will need the notion of a negligible function, which intuitively is simply a function that approaches zero very fast:

Definition 3 (Negligible Function)**.** *Let $\mu : \mathbb{N} \to \mathbb{R}$ be a function. We say that μ is negligible if for every positive polynomial $q > 0$, there exists some $x_0 \in \mathbb{N}$ such that for all $x > x_0$, the following holds:*

$$|\mu(x)| < \frac{1}{q(x)}.$$

In other words, μ approaches zero faster than any inverse positive polynomial.

Informally, our second notion of comparing ensembles roughly states that as we increase the security parameter λ, the ensembles quickly approach one another:

Definition 4 (Statistical Closeness)**.** *Let A, B be two probability distributions over some countable domain D. The statistical distance between A and B is defined as follows:*

$$\Delta(A, B) = \frac{1}{2} \sum_{\alpha \in D} \left| \Pr_{a \leftarrow A}[a = \alpha] - \Pr_{b \leftarrow B}[b = \alpha] \right|.$$

Let $X = \{X_\lambda\}_{\lambda \in \mathbb{N}}$ and $Y = \{Y_\lambda\}_{\lambda \in \mathbb{N}}$ be two ensembles. We say that X and Y are statistically close if the following function is negligible:

$$\mu(\lambda) = \Delta(X_\lambda, Y_\lambda).$$

If X and Y are statistically close, we write $X \approx Y$.

Exercise 1. *Consider the following two random processes:*

- *The first flips λ fair coins and outputs 1 if all coins come up heads; else it outputs 0.*
- *The second always outputs 0.*

What can we say about the ensembles that describe the outputs of these two processes?

Our final and weakest notion of comparing ensembles does not state that the ensembles are actually close together, merely that no bounded entity can tell they are not close together.

Definition 5 (Computational Indistinguishability)**.** *Let $X = \{X_\lambda\}_{\lambda\in\mathbb{N}}$ and $Y = \{Y_\lambda\}_{\lambda\in\mathbb{N}}$ be two ensembles. We say that X and Y are computationally indistinguishable (or often just indistinguishable) if for every (non-uniform) probabilistic polynomial-time algorithm D, the following function is negligible:*

$$\mu(\lambda) = \left| \Pr_{x\leftarrow X_\lambda}[D(x) = 1] - \Pr_{y\leftarrow Y_\lambda}[D(y) = 1] \right|.$$

Informally, as we increase λ, no D can reliably determine if it is seeing a sample from X_λ or from Y_λ. If X and Y are indistinguishable, we write $X \stackrel{c}{=} Y$.

Note that the following relationships hold for ensembles X, Y:

$$X \equiv Y \implies X \approx Y \qquad \text{and} \qquad X \approx Y \implies X \stackrel{c}{=} Y.$$

That is, equivalence is our strongest notion of closeness between ensembles, and computational indistinguishability is our weakest.

Exercise 2. *Prove the above relationships hold.*

Remark 1. *Very often, we will leave the security parameter λ formally implicit. Namely, we will not necessarily mark every ensemble with an explicit symbol λ. This is simply to reduce noisiness of our written mathematics. When a security parameter is needed, you may simply assume that every ensemble/procedure is indexed by it.*

Two-Party Semi-Honest Security

Now, we are ready to define our first notion of protocol security. For simplicity, we will start by defining security for two-party protocols.

Later in this course, we will consider security in the presence of parties who arbitrarily deviate from the protocol as we describe it. We say such a party is malicious or active. For now, however, we will consider a weaker notion of security called semi-honest security (or sometimes passive security). In this notion, the parties precisely follow the protocol as we describe it, and they try to learn as much as possible by observing protocol messages. That is, our considered adversary is very weak.

There are three very good reasons to consider this weak notion of security:

- **Simplicity.** It is much simpler to describe and analyze semi-honest-secure protocols than maliciously-secure protocols. Thus, for pedagogical reasons it is convenient to start from this simpler definition.
- **Useful.** Semi-honest security captures the notion of an ex post facto attack. Suppose our two parties are honest during a protocol execution, but one of the parties becomes corrupted long after the protocol is finished. For instance, perhaps that party is the victim of a data breach. Semi-honest security provides a provable guarantee in this scenario where any logs kept by the corrupted party leak nothing about the other party's input.
- **Upgradability.** As we will see in this course, most of our techniques for achieving malicious security are constructed as follows: (1) Construct a protocol secure against a passive adversary. (2) Upgrade that protocol to work against an active adversary. Thus, studying passive security is a natural first step in understanding active security.

Now, recall the notion of a real world / ideal world paradigm. In the real world, the parties observe protocol messages, and in the ideal one, they simply observe the function input/output behavior. We will define security by comparing these two quantities.

Let's start by formalizing what a party sees in the real world:

Definition 6 (View)**.** *Let* Π *be a protocol. The view of a party* P *during execution of* Π *is a procedure that describes the information observed by* P*. We denote this procedure by* $\texttt{View}_P^\Pi$*. In particular,* $\texttt{View}_P^\Pi$ *takes as input all party inputs, and its output describes an ensemble containing:*

- *P's input to* Π*.*
- *Any randomness sampled by* P *during execution of* Π*.*
- *All messages received by* P *during execution of* Π*.*

Definition 7 (Two Party Semi-Honest Security (for deterministic computations))**.** *Let* f *be a computable function. Let* A, B *be parties with respective inputs* x, y*. We say that a protocol* Π *securely computes* f *in the presence of a semi-honest adversary when (1) the protocol is* ***correct*** *(i.e., the parties output* $f(x, y)$ *except with negligible probability in* λ*) and (2) for each party, there exists a probabilistic polynomial time algorithm* $\mathcal{S}_A$ *(resp.* $\mathcal{S}_B$*)—called a* ***simulator****—such that for all inputs* x, y*, the following hold:*

$$\texttt{View}_A^\Pi(x, y) \stackrel{c}{=} \mathcal{S}_A(x, f(x, y)) \qquad \texttt{View}_B^\Pi(x, y) \stackrel{c}{=} \mathcal{S}_B(y, f(x, y)).$$

If the above symbols $\stackrel{c}{=}$ *are replaced by* $\approx$*, we say that* Π *has statistical security. If the above symbols* $\stackrel{c}{=}$ *are replaced by* $\equiv$*, we say that* Π *has perfect security.*

A small remark is that the above definition is specifically for deterministic (i.e., non-randomized) computations only. There is some nuance in handling randomized computations, which we will discuss later.

The above definition precisely captures the intuition discussed at the beginning: given idealized access to f, each party can make up a view that looks like the real world view. The process of making up such a view is called a simulator.

Note that to prove a protocol is secure, a cryptographer must go through a creative step: they must design simulators that show it is secure. Sometimes, constructing a simulator is easy and sometimes it is not.

First Proof of Security

Let's work on showing Protocol 1 is secure. First, let's describe the view of each party. Note that in contrast to constructing a simulator, constructing the view is not a creative process; it is merely a mechanical accounting of the information the parties see as a consequence of Π. In our example protocol, the view is trivial: each party sees its own input, and it sees the message sent by the other party.

$$\texttt{View}_A^\Pi(x, y) = \{\ (x, y)\ \} \qquad \texttt{View}_B^\Pi(x, y) = \{\ (y, x)\ \}.$$

Now, let's construct a simulator for A (the simulator for B is symmetric). Recall that the simulator $\mathcal{S}_A$ is given x and the output $x \oplus y$, and its job is to compute something that looks like $\texttt{View}_A^\Pi$. Let us start with a simulator that does not work:

$$\texttt{BadS}(x, x \oplus y) = \{\ (x, r)\ \mid r \leftarrow \{0, 1\}\ \}.$$

At first glance, this attempt might seem sensible. This simulator tries to model B's bit y by sampling a uniform bit r. However, this is not a good simulation. This is because B's input y is, in general, not random. Indeed, our goal as protocol designers is to design a protocol that works for any user, so it is not safe for us to make an assumption about how the user's input might be distributed. Formally, y is universally quantified, not probabilistic. The takeaway is this:

> In general, protocol inputs are not random, and we typically do not make assumptions about how inputs are distributed.

To conclude, let's construct a good simulator for A:

$$\mathcal{S}_A(x, x \oplus y) = \{\ (x, x \oplus (x \oplus y))\ \}.$$

The proof that this is a good simulation is straightforward:

$$\mathcal{S}_A(x, x \oplus y) = \{\ (x, x \oplus (x \oplus y))\ \} \equiv \{\ (x, y)\ \} = \mathtt{View}_A^{\Pi}(x, y).$$

Thus, Protocol 1 is perfectly secure in the presence of a semi-honest adversary. The simulator succeeds because the output $x \oplus y$ together with x reveals y, so the protocol leaks no more information than the ideal functionality itself.

Exercise 3. *Last time, we saw a p-party protocol for computing the sum of p values by passing messages in a circle with a mask chosen uniformly from* $[0, ..., 2^6 0)$. *Consider modifying the protocol such that the mask is chosen uniformly from* $[0, ..., 2^\lambda)$. *Now, the security definition above naturally generalizes to p parties, where no two parties collude (we will see how to handle collusions later). In particular, one must show a simulator for each of the p parties. Construct simulators and argue that the p-party protocol is statistically secure in the above sense.*

Next Time

We will expand our definitions, and we will see our first non-trivial cryptographic techniques for achieving secure computation.