Randomized Functionalities and Semi-Honest Composition

CS 507, Topics in Cryptography: Secure Computation

David Heath

Fall 2026

Last time, we introduced the notion of 1-out-of-2 oblivious transfer (OT), and we built on the ideal OT functionality to achieve a protocol that deals a multiplication triple. We then used the triple functionality to achieve general 2PC for deterministic functions, via the GMW protocol. What we did, then, is built up the GMW protocol by means of *composition*. We defined simple functionalities, then used those functionalities to build protocols that realize richer functionalities, until ultimately reaching our goal.

In more detail, what we technically did is as follows:

- We defined an ideal functionality for OT $\mathcal{F}_{\texttt{OT}}$.
- We devised a protocol $\Pi_{\texttt{OT}}$ (based on DDH) that realizes $\mathcal{F}_{\texttt{OT}}$.
- We defined an ideal functionality for dealing a multiplication triple $\mathcal{F}_{\texttt{triple}}$.
- We devised a protocol $\Pi_{\texttt{triple}}$ where *parties call* $\mathcal{F}_{\texttt{OT}}$ *as a black box* and that realizes $\mathcal{F}_{\texttt{triple}}$.
- We defined an ideal functionality for secure 2PC of arbitrary deterministic functions $\mathcal{F}_{\texttt{2PC}}$.
- We devised a protocol $\Pi_{\texttt{GMW}}$ where *parties call* $\mathcal{F}_{\texttt{triple}}$ *as a black-box* and that realizes $\mathcal{F}_{\texttt{2PC}}$.

Thus, the protocol descriptions for $\Pi_{\texttt{triple}}$ and $\Pi_{\texttt{GMW}}$ are, in fact, not for real-world protocols. These protocols instead live in some *hybrid world*, neither the real world nor the ideal world, but rather some world in between, where the parties do not have access to a trusted third party computing the function they care about, but rather a trusted third party solving some *easier* problem.

The point of stating that a cryptographic protocol realizes a particular functionality is that we should be able to *replace* the calls to the ideal functionality by calls to the underlying protocol, and the resulting *composed* protocol should be secure. However, technically we have not seen that this is true.

Today, we will clean up our understanding of ideal functionalities, as well as refine our definition of semi-honest security, ultimately with the goal of obtaining a clear picture of semi-honest secure protocols, including in terms of how they compose. Our particular goals are as follows:

- Understand why our formal definition of security has been so far restricted to deterministic functionalities.
- Define security for randomized functionalities.
- Generalize our definition to the *dishonest majority setting*, where the adversary corrupts any number of parties.
- Formalize semi-honest protocol composition.

Semi-honest Security: Randomization and Multiple Parties

Recall that our working definition for semi-honest security is as follows:

Definition 1 (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 if for each party, there exists a probabilistic polynomial time algorithm $\texttt{Sim}_A$ (resp. $\texttt{Sim}_B$)—called a simulator—such that for all inputs x, y, the following hold:*

$$\texttt{View}_A^{\Pi}(x, y) \stackrel{c}{=} \texttt{Sim}_A(x, f(x, y))$$
$$\texttt{View}_B^{\Pi}(x, y) \stackrel{c}{=} \texttt{Sim}_B(y, f(x, y))$$

There are two shortcomings of this definition:

- It only defines security for two parties. We want a definition that explains security for any number of parties.
- It only defines security for protocols that compute *deterministic* output. It can also be desirable to construct protocols that generate *randomized* outputs.

Our goal now is to construct a more general definition of semi-honest security that addresses these shortcomings.

Let's start by understanding why the definition does not work for randomized functionalities. First, note that it is desirable to have a definition that works for randomized functionalities. Indeed, we have already informally assumed such a notion is well-defined! The multiplication triple interface we informally introduced last time is clearly a randomized functionality, since it involves the third party flipping coins.

Now, let's see why our working definition is incompatible with randomized functionalities. To understand this, let's define a randomized functionality and then (erroneously) apply Definition 1. We will see using Definition 1 will allow us to construct protocols that are "secure", but that clearly defy our intuition of what *should* be considered secure.

To see this, let's revisit a standard notion from cryptography called a *pseudorandom generator* (PRG):

Definition 2 (Pseudorandom Generator)**.** *An efficiently computable deterministic function $G : \{0,1\}^\lambda \to \{0,1\}^{2\lambda}$ is considered a **pseudorandom generator** if the following are indistinguishable:*

$$\left\{ r \;\middle|\; \begin{array}{l} s \leftarrow \{0,1\}^\lambda \\ r := G(s) \end{array} \right\} \stackrel{c}{=} \left\{ r \;\middle|\; r \leftarrow \{0,1\}^{2\lambda} \right\}$$

Intuitively, a pseudorandom generator produces long strings that look truly random from the perspective of an adversary that does not know the short seed s.

Now let's invent a possibly useful functionality:

Functionality 1. *The functionality interacts with two parties A and B.*

- ***Parameters.*** *Parties agree on a PRG G.*
- ***Input.*** *Parties have no input.*
- ***Output.*** *A outputs a random PRG seed $s \leftarrow \{0,1\}^\lambda$. B outputs $G(s)$.*

Note that this functionality is *randomized*, because it outputs a random seed to A. Thus, Definition 1—which is well-defined only for deterministic functionalities—inherently cannot be used here. However, let's try using it anyway, to see where things go wrong.

The problem is that our definition of security does not yet provide any insistence that the party's simulated view be consistent with the functionality output. In particular, it's possible to "prove security" of the following obviously insecure protocol:

- A uniformly samples a seed $s \leftarrow \{0,1\}^\lambda$.
- A sends s to B and outputs s.
- B locally computes and outputs $G(s)$.

Indeed, our intuition is that this protocol should be insecure, since the functionality does not say that B is permitted to learn s, but clearly he does.

Exercise 1. *Construct simulators for A and B that "prove" this protocol secure in the context of Definition 1.*

To fix this, we will upgrade our definition of semi-honest security to require that the simulation must make sense in the context of the outputs of *every* party:

Definition 3 (Two Party Semi-Honest Security)**.** *Let f be a (possibly-randomized) functionality. 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 if for each party, there exists a probabilistic polynomial time algorithm $\texttt{Sim}_A$ (resp. $\texttt{Sim}_B$)—called a simulator—such that for all inputs x, y, the following hold:*

$$\left\{ \left(\texttt{View}_A^\Pi(\tau), \texttt{Output}^\Pi(\tau)\right) \mid \tau \leftarrow \Pi(x,y) \right\} \stackrel{c}{=} \left\{ \left(\texttt{Sim}_A(x, z_0), (z_0, z_1)\right) \mid (z_0, z_1) \leftarrow f(x,y) \right\}$$
$$\left\{ \left(\texttt{View}_B^\Pi(\tau), \texttt{Output}^\Pi(\tau)\right) \mid \tau \leftarrow \Pi(x,y) \right\} \stackrel{c}{=} \left\{ \left(\texttt{Sim}_B(y, z_1), (z_0, z_1)\right) \mid (z_0, z_1) \leftarrow f(x,y) \right\}$$

Above, $\tau \leftarrow \Pi(x,y)$ denotes running the protocol Π with inputs x, y yielding a complete protocol transcript τ; the protocol transcript τ includes all inputs, all messages exchanged, and all coins tossed, by all parties. The functions `View` *and* `Output` *respectively fetch from the transcript (1) the specified party view and (2) the concatenation of all party outputs.*

Note that the key adjustment to the definition is that each party's view must be simulated in the context of a particular functionality output. This rules out our above trivial protocol.

Exercise 2. *Prove that for the above trivial PRG protocol and for Definition 3, there is no simulator $\texttt{Sim}_B$ for B.* Hint: You can use any such $\texttt{Sim}_B$ to break the security property of the PRG G in polynomial time, which is assumed to be impossible.

Now, let's upgrade our definition to also handle functionalities that involve any number of parties. The solution here is straightforward: We must construct simulators for *every* proper subset of parties:

Definition 4 (Semi-Honest Security)**.** *Let f be a (possibly-randomized) n-party functionality running between parties $P_0, ..., P_{n-1}$. Let P_i have input x_i. We say that a protocol Π securely computes f in the presence of a semi-honest adversary if for each proper subset of parties $C \subset \{0, ..., n-1\}$, there exists a probabilistic polynomial time algorithm $\texttt{Sim}_C$—called a simulator—such that for all inputs $x_0, ..., x_{n-1}$, the following hold:*

$$\left\{ \left(\texttt{View}_C^\Pi(\tau), \texttt{Output}^\Pi(\tau)\right) \mid \tau \leftarrow \Pi(x_0, ..., x_{n-1}) \right\}$$
$$\stackrel{c}{=} \left\{ \left(\texttt{Sim}_C\left(x_C, y_C\right), (z_0, ..., z_{n-1})\right) \mid (z_0, ..., z_{n-1}) \leftarrow f(x_0, ..., x_{n-1}) \right\}$$

Above, $\texttt{View}_C^\Pi$ denotes the concatenation of views of all corrupted parties, and similarly x_C (resp. z_C) denotes the concatenation of all corrupted party inputs (resp. outputs).

Exercise 3. *(The following is also a homework problem.) The above definition requires us to simulate all proper subsets of parties. Consider the 3-party setting. One might naively assume that it suffices to simulate each* size-two *subset of corrupted parties, without needing to simulate subsets of size one. Show why this is not sufficient, i.e. that there are protocols that are (1) secure when you only consider subsets of size two but (2) insecure when you consider all subsets.*

Protocol Composition

Now, let's return to the problem introduced at the start of this lecture. We built up protocols in a modular way, using the ideal functionality as an *interface* between protocols. Namely, we have some protocol Π, say our triple protocol, which is built *in terms of* some second protocol ρ, say our OT protocol.

Here's one ways we might prove security of Π:

> **The monolithic approach.** To prove security of Π, we inline the code of all calls to ρ, then provide simulators that describe both the view internal to calls to ρ, and the view in parts of Π that are not calls to ρ.

This approach would technically work, and indeed it even has the advantage that the approach is extremely simple—we don't need any new tools to take this approach!

On the other hand, this monolithic approach has serious and glaring flaws.

- First, the proof of security for Π is not "reusable". In particular, suppose that tomorrow, someone devises a better 1-out-of-2 OT protocol σ. If we wish to reimplement Π in terms of σ instead of ρ, we will need a new proof of security – after all, many of the messages sent in the protocol will have changed!

- Second, the proof of security for Π is "complex". When writing the monolithic proof of Π's security, we must simultaneously keep in mind what the party sees as a result of Boolean gate handling, and as a result of OT messages (e.g., DDH group elements). This makes writing the proof harder and more error-prone.

What would be preferable is clearly a **modular approach** to security. We would like to reason about Π without concerning ourselves with the details of ρ. In particular, when we explained our triple protocol, we used OT in a black-box manner, making statements like "the parties call 1-out-of-2 OT". Thus our description of Π is modular, and we would like our proof of security to *inherit* that modularity.

Fortunately, it is possible to think in this way. Namely, there exists a relatively simple *composition theorem* for semi-honest protocols.

Recall that we define security of protocols with respect to ideal functionalities. I.e., perhaps protocol ρ securely instantiates some functionality g, and we would like to prove that Π—which calls ρ—securely instantiates some functionality f. The composition theorem states that it is sufficient for us to think of Π as calling ρ's ideal functionality g, and composed security falls out for free. This is exactly what we want! We can reason about 1-out-of-2 OT as if it is a black-box functionality.

The protocol Π written in terms of ideal g is sometimes said to be formalized in a *hybrid model*. For instance our triple protocol is in the OT-hybrid model: it assumes OT is implemented as an ideal functionality. A party's view in the g-hybrid model includes its inputs to and outputs from each call to g, but nothing else about g's execution.

When we instantiate the black-box g with a real-world protocol ρ, security is already proven. Moreover, if tomorrow we find a new secure OT protocol σ, we can substitute σ instead, and no new proof of security is necessary.

Theorem 1 (Semi-honest protocol composition)**.** *Let Π be a protocol in the g-hybrid model (i.e. Π makes calls to the ideal functionality g). Suppose Π realizes ideal functionality f. Let ρ be a protocol that realizes g. Then $\Pi[\rho/g]$ realizes f. Namely, if we replace all calls to ideal functionality g by protocol ρ, then the resulting composed protocol also securely computes f.*

Proof Sketch. Recall that semi-honest security requires existence of a simulator. Thus (1) to show this theorem, we must construct a simulator for every proper subset of corrupted parties, and (2) we can assume that we *already have* simulators (for the same subset of parties) for both Π (written in terms of g) and ρ. Let's call those simulators S_Π and S_ρ. Our goal is to construct a simulator $S_{\Pi[\rho/g]}$.

In short, the simulator $S_{\Pi[\rho/g]}$ is straightforward to construct. $S_{\Pi[\rho/g]}$ first calls S_Π to produce a view, including interactions involving ideal functionality g. In particular, we can extract from this simulated view the input/output behavior of each interaction with g. Let the input (resp. output) be x_C (resp. z_C). For each such call to g, we can call $S_\rho(x_C, z_C)$ to obtain a view corresponding to real-world protocol ρ. We now replace the part of the view corresponding to g with newly generated view of ρ. Once all such calls to g are replaced, we have our simulator $S_\Pi[\rho/g]$. This simulator's output is indeed indistinguishable from the real-world view; this can be shown formally by arguing that, if it is not indistinguishable, then one can distinguish the real-world view of ρ from the output of simulator S_ρ, a contradiction.

For a careful proof, see [Gol01] Theorem 7.3.3. □

By using this theorem, we can easily prove security of our OT-based 2PC protocol:

Lemma 1. *Let $\Pi_{\texttt{GMW}}$ be the 2PC protocol described in Lecture 5. $\Pi_{\texttt{GMW}}$ is perfectly semi-honest secure in the OT-hybrid model.*

Note that we can claim Π is perfectly secure. Of course, we lose perfect security once we instantiate the OT functionality by a particular computationally-secure protocol. This is implied by the protocol composition theorem.

Exercise 4. *Write out simulators for $\Pi_{\texttt{GMW}}$ in the OT-hybrid model model. Check that you understand how the composition theorem will apply to those simulators.*

Next Time

So far, all of our protocols incur high *round complexity*. These protocols are based on an interactive trick involving XOR secret sharing of bits. Next time, we will begin studying a different route to MPC protocols called the *Garbled Circuit* (GC) technique. GC has the amazing property that, regardless of the desired computation, we can construct protocols that run in a small constant number of rounds of communication. The trade-off will be an increase in overall bandwidth consumption.

References

[Gol01] Oded Goldreich. *Foundations of cryptography: volume 2, basic applications*, volume 2. Cambridge university press, 2001.