Oblivious Transfer

CS 507, Topics in Cryptography: Secure Computation

David Heath

Fall 2025

Last time, we saw our first general-purpose (i.e., one that works for any bounded computable function f) MPC protocol. Recall that the protocol worked between three semi-honest parties A, B, and D, where we assumed no two parties collude. In short, the protocol proceeds as follows:

- First, the parties encode f as a Boolean circuit $C : \{0,1\}^n \to \{0,1\}^m$. This is possible for any bounded computable function f. Recall that we set up C such that it consists of INPUTs, OUTPUTs, ANDs, XORs, and NOTs.
- Next, the parties proceed gate by gate through C, obtain that A and B hold XOR secret shares of the real value on each wire in C.
- The main non-triviality of the protocol occurs at AND gates. To handle an AND gate, we can assume A and B hold secret shares of the input wires $[a]$ and $[b]$, and our goal is to compute $[a \cdot b]$. Recall that to achieve this, A and B make use of the dealer D, who is asked to deal a uniform Beaver triple:

  $$\{ \;\; [\alpha], [\beta], [\alpha \cdot \beta] \quad \text{where} \quad \alpha, \beta \in_\$ \{0,1\} \; \}$$

  Given this triple, A and B can exchange a few messages and perform a few local operations to securely obtain $[a \cdot b]$.

This protocol is surprisingly performant, especially in terms of communication complexity and computation. However, we discussed that the protocol also has several serious shortcomings.

Today, we will address one of these shortcomings: we will remove the dealer D. This is clearly desirable, since in general our party A (resp. B) might not trust that B (resp. A) is not working together with D. If two parties do, in fact, collude, then the privacy of the remaining party is compromised.

Exercise 1. *Show how in the context of our protocol from last time, there exist circuits C where parties B and D can jointly learn A's entire input x, even when x is not implied by the output of f. Explain how B and D learn x.*

Today we will show how to upgrade our protocol such that it will work for two parties A and B, and we can remove the need for a dealer D. Note that we will still, for now, be working in the semi-honest model.

The main idea for removing the dealer D will be to emulate D's existence by means of a cryptographic protocol. Indeed, our main task question today is as follows: How can parties A and B construct a uniform Beaver triple $[\alpha], [\beta], [\alpha \cdot \beta]$ without either of them learning α or β? To achieve this, we will introduce cryptographic primitive that is central to the study of secure protocol design: oblivious transfer (OT). In particular, we will achieve goals:

- We will define OT.
- We will see how to use OT.
- We will see how to construct OT.

The OT Functionality

An oblivious transfer (OT) protocol is a cryptographic protocol involving two parties: a sender S and a receiver R. Roughly, OT works as follows. The sender S has two secret messages m_0 and m_1. Assume that the receiver R does not know the content of these messages, but she does know that there are two messages, and that S has arranged them in the order m_0, m_1. The receiver R would like to learn one of these messages. Namely, the receiver R has a selection bit b, and the goal of the protocol is to deliver (only) m_b to R.

We can make this more formal by specifying the *ideal functionality* for OT:

Functionality 1 (1-out-of-2 Oblivious Transfer)**.** *The functionality interacts with two parties S and R.*

- ***Parameters.*** *Parties agree on the length of messages n.*
- ***Input.*** *S inputs two messages $m_0, m_1 \in \{0,1\}^n$; R inputs a choice bit $b \in \{0,1\}$.*
- ***Output.*** *R outputs m_b. S outputs nothing (which we sometimes denote by $\perp$).*

The above functionality is somewhat abstract, and it can be hard to understand. For instance, if R knows nothing about m_0 and nothing about m_1, how is it that she knows which one she wants to receive? In short, OT is typically used not as a "top-level" protocol, but rather as a simple building block by which to construct more complex protocols. In such scenarios, we can ensure R knows which message she wants, even though she doesn't know the content of the messages themselves; this will become clearer when we use OT to construct larger protocols.

Above, we specified OT as an ideal functionality. Recall that an ideal functionality is a specification that details precisely what parties are allowed to learn by running a protocol. So a secure OT protocol ensures that the only information conveyed is that R learns m_b. In particular, the protocol ensures that:

- R learns nothing about "other message" m_{1-b}.
- S learns nothing about R's choice bit b.

How to Use OT

As a warm-up, let's show that OT can be used to solve non-trivial problems. In particular, let's solve the secure AND problem from a few lectures ago:

Functionality 2 (Secure AND)**.** *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 $x \cdot y$.*

Recall that before we used a dealer to solve this problem; now we can solve it by invoking OT:

Protocol 1.

- *We will invoke the OT functionality, so we need to assign OT roles to our parties. Let's say that A plays the sender S and B plays the receiver R (the other way can also, of course, be made to work).*
- *Since A is the sender, she must select two messages m_0 and m_1. In this case, we will set $m_0 = 0$ and $m_1 = x$.*
- *Since B is the receiver, he must select a choice bit b. We will set $b = y$.*
- *Now, the parties invoke the OT functionality with their inputs. By the definition of OT, B receives the following:*

$$m_b = m_y = \left(\begin{cases} m_0 & \textit{if } y = 0 \\ m_1 & \textit{if } y = 1 \end{cases}\right) = \left(\begin{cases} 0 & \textit{if } y = 0 \\ x & \textit{if } y = 1 \end{cases}\right) = \left(\begin{cases} x \cdot y & \textit{if } y = 0 \\ x \cdot y & \textit{if } y = 1 \end{cases}\right) = x \cdot y$$

Thus, B indeed receives $x \cdot y$.

- *To complete the protocol, we need to arrange that A also learns $x \cdot y$. Since we are in the semi-honest setting, this is easy to arrange: B sends the output $x \cdot y$ to A.*

So we can indeed solve the secure AND problem with a single invocation of OT.

Now, let's work on a slightly harder problem: let's show how to construct a Beaver triple from two calls to OT:

Protocol 2. *The protocol runs between two parties A and B who wish to construct a uniform, secret-shared Beaver triple.*

- ***Input.*** *Neither party has input.*
- ***Output.*** *Parties output uniform secret shares $[\alpha], [\beta], [\alpha \cdot \beta], where \alpha, \beta \in_\$ \{0, 1\}$ are uniform.*
- ***Procedure.***

  - *Our first goal will be to sample uniform secret shares $[\alpha]$ and $[\beta]$. This turns out to be easy: A uniformly samples $\alpha_0, \beta_0 \in_\$ \{0, 1\}$, and B uniformly samples $\alpha_1, \beta_1 \in_\$ \{0, 1\}$. Notice that these shares implicitly define α and β:*

    $$\alpha \triangleq \alpha_0 \oplus \alpha_1$$
    $$\beta \triangleq \beta_0 \oplus \beta_1$$

    Thus, parties hold $[\alpha], [\beta]$.
  - *Now, our next goal is to compute $[\alpha \cdot \beta]$. To do so, let's observe the following:*

    $$\alpha \cdot \beta = (\alpha_0 \oplus \alpha_1) \cdot (\beta_0 \oplus \beta_1) \qquad = \alpha_0 \cdot \beta_0 \oplus \alpha_0 \cdot \beta_1 \oplus \alpha_1 \cdot \beta_0 \oplus \alpha_1 \cdot \beta_1$$

    Thus, our goal is to compute all four summands above, in a secret-shared form. If we can, we can complete the protocol.
  - *First, notice that A can locally compute $\alpha_0 \cdot \beta_0$, since she knows both $\alpha_0 and \beta_0$. Similarly, B can compute $\alpha_1 \cdot \beta_1$. The main challenge is to compute the cross terms $\alpha_0 \cdot \beta_1$ and $\alpha_1 \cdot \beta_0$, where each party knows only one of the multiplicands.*
  - *Let's focus on computing $\alpha_0 \cdot \beta_1$ ($\alpha_1 \cdot \beta_0$ will be computed symmetrically). Notice that Protocol 1 can (almost) be used here! However, it turns out that it is not quite secure for the parties to learn $\alpha_0 \cdot \beta_1$ directly, since this intermediate term is not implied by the protocol output (i.e., it is not possible to simulate this quantity). Thus, we will instead make the following small adjustment to Protocol 1 which causes parties to compute secret shares $[\alpha_0 \cdot \beta_1]$, rather than $\alpha_0 \cdot \beta_1$ in the clear: A will act as OT sender. She samples a uniform bit r and sets $m_0 = r, m_1 = r \oplus \alpha_0$. B acts as OT receiver and sets $b = \beta_1$. B thus receives m_b, which basic reasoning reveals is equal to $r \oplus \alpha_0 \cdot \beta_1$. Notice that $(r, r \oplus \alpha_0 \cdot \beta_1)$ form an XOR secret share of $\alpha_0 \cdot \beta_1$.*
  - *Parties symmetrically compute the following: $(s, s \oplus \alpha_1 \cdot \beta_0)$, where s is uniform and chosen by A.*
  - *From here, parties essentially add up what they have seen, and the result is an XOR sharing $[\alpha \cdot \beta]$. More precisely, notice that A now knows the following quantities:*

    $$\alpha_0 \cdot \beta_0 \qquad \textit{and} \qquad s \qquad \textit{and} \qquad r$$

    Thus, A can compute the following sum:

    $$\alpha_0 \cdot \beta_0 \oplus s \oplus r$$

and B now knows the following quantities:

$$\alpha_1 \cdot \beta_1 \qquad \textit{and} \qquad r \oplus \alpha_0 \cdot \beta_1 \qquad \textit{and} \qquad s \oplus \alpha_1 \cdot \beta_0$$

Thus, B can compute the following sum:

$$\alpha_1 \cdot \beta_1 \oplus r \oplus \alpha_0 \cdot \beta_1 \oplus s \oplus \alpha_1 \cdot \beta_0$$

It's easy to see that if the two party sums form a sharing $[\alpha \cdot \beta]$.

– *Parties output their respective shares in $[\alpha], [\beta], [\alpha \cdot \beta]$.*

Exercise 2. *Think about how you would construct simulators for Protocol 2. We technically have not yet learned how such simulators would work, since Protocol 2 involves invoking another protocol (namely OT) as a black-box. However, you can reflect on what the rules of such a simulation should be.*

Now that we have Protocol 2, we can construct our first general-purpose two-party semi-honest-secure protocol: The parties just run the same dealer-assisted protocol from last time, but whenever they need a triple, they invoke Protocol 2.

How to Construct OT

Now that we have seen how an OT protocol can be useful, let's try to actually build such a protocol. It turns out that—for the first time in this course—we are confronted with solving a problem that is in some sense fundamentally hard. In particular, it is known that an Oblivious Transfer protocol requires[1] a public-key encryption scheme. This is in stark contrast to all the other techniques we have seen so far this course, which have all been constructed from basic information-theoretic techniques.

Note that there are many ways to construct OT. Here, we will investigate a very explicit implementation of semi-honest 1-out-of-2 OT. The goal is to make the technique as concrete as possible, such that nothing is left to the imagination.

In particular, we will leverage a computational hardness problem called the Decisional Diffie Hellman (DDH) assumption to construct OT:

Definition 1 (Finite Cyclic Group)**.** *A **finite group** G is a finite set of elements (often also denoted G) together with an operation $\cdot : G \times G \to G$ s.t. the following hold:*

- *The operation $\cdot$ is associative and has an identity element 1_G.*
- *Inverses: For every element $h \in G$, there exists an element $h^{-1} \in G$ s.t. $h \cdot h^{-1} = 1_G$ and $h^{-1} \cdot h = 1_G$.*

*The **order** of G is the number of elements in G. A group G is **cyclic** if there exists some element $g \in G$, called the **generator**, such that every element $h \in G$ can be expressed as g^a where $a \in \mathbb{Z}_q$ is a number. The notation g^a indicates that g is multiplied with itself a times.*

Definition 2 (Decisional Diffie Hellman Assumption)**.** *Let G_λ denote a family of finite cyclic groups indexed by a security parameter λ. Let group $G = G_\lambda$ have order q, and let g be the generator of G. The **Decisional Diffie Hellman** problem is hard over family G_λ if the following (families of) are indistinguishable:*

$$\{ \quad g^a, g^b, g^{a \cdot b} \quad \textit{where} \quad a, b \in_\$ \mathbb{Z}_q \quad \} \overset{c}{=} \{ \quad g^a, g^b, g^c \quad \textit{where} \quad a, b, c \in_\$ \mathbb{Z}_q \quad \}$$

It is believed that there are certain families of finite cyclic groups where DDH is hard (for classical adversaries), such as certain families of elliptic curve groups.

Now, let's DDH to construct OT. Very roughly the protocol will proceed by observing it is easy to construct public-key encryption from DDH. In particular, the protocol sketch is as follows:

[1]Actually, the formal statement is more nuanced than this. Take a look at [IR90] for more details.

- The OT receiver R will generate and send to S two public keys $\texttt{pk}_0$ and $\texttt{pk}_1$. However, only *one* of these keys will have a corresponding secret key.
- The OT sender will encrypt message m_i with the corresponding public key $\texttt{pk}_i$ and send the encryptions to R.
- Since R only has one secret key, she can only decrypt one message.

More formally:

Protocol 3 (DDH-Based Semi-honest OT)**.** *Let us assume that S's messages m_0, m_1 come from a DDH group G of order q with generator g. R has choice bit b.*

- *R uniformly samples $a \in_\$ \mathbb{Z}_q$. She defines two "public keys": $h_b = g^a$, and $h_{1-b} \in_\$ G$. I.e., h_b has a corresponding "secret key" a, but h_{1-b} does not. R sends h_0, h_1 to S.*
- *S samples two random values $r_0, r_1 \in_\$ \mathbb{Z}_q$. These are needed to encrypt.*
- *S sends the following quantities to R:*

$$g^{r_0} \qquad \text{and} \qquad g^{r_1} \qquad \text{and} \qquad h_0^{r_0} \cdot m_0 \qquad \text{and} \qquad h_1^{r_1} \cdot m_1$$

- *R decrypts m_b as follows:*

$$\frac{h_b^{r_b} \cdot m_b}{(g^{r_b})^a} = \frac{(g^a)^{r_b} \cdot m_b}{(g^{r_b})^a} = \frac{(g^{ar_b}) \cdot m_b}{(g^{ar_b})} = m_b$$

There are two claims to be proved:

- S's view can be simulated. This is easy to show, as all S receives are two uniformly random elements from G.
- R's view can be simulated. This is true, under the assumption that DDH holds. In particular, the message $h_{1-b}^{r_{1-b}} \cdot m_{1-b}$ looks uniform to R.

Exercise 3. *Formally work out the simulation for R.*

Next Time

Next time, we will continue to explore OT as a primitive. In addition, we've today saw our first instance of protocol composition. In particular, we've seen how to build one protocol in terms of another, simpler protocol. Technically, we do not yet have any formal notion of security for this composition. We will soon see will how semi-honest security works in a compositional way.

References

[IR90] Russell Impagliazzo and Steven Rudich. Limits on the provable consequences of one-way permutations. In Shafi Goldwasser, editor, *CRYPTO'88*, volume 403 of *LNCS*, pages 8–26. Springer, New York, August 1990.