Oblivious Transfer Continued; Randomized Functionalities

CS 507, Topics in Cryptography: Secure Computation

David Heath

Fall 2025

Last time, we introduced the 1-out-of-2 oblivious transfer (OT) functionality:

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 $\perp$).*

Recall that this functionality is a fundamental building block in MPC. In particular, we saw how to use an OT to allow two parties to generate a secret-shared Beaver triple. From this, we were able to construction our first semi-honest secure 2PC protocol.

Today, we will make concrete our understanding of OT by constructing an OT protocol and proving it secure. Then, we will return to our fundamentals of security and upgrade our notion of semi-honest security to handle any number of parties, as well as possibly-randomized functionalities.

Recap of How to Achieve OT

Recall that we sketched one way of how to construct semi-honest 1-out-of-2 OT. In particular, we looked at how to build OT from (certain kinds of) public key encryption schemes $(\texttt{KeyGen}, \texttt{Enc}, \texttt{Dec})$:

- The receiver R generates two public keys as follows:

$$(\texttt{pk}_b, \texttt{sk}_b) \leftarrow \texttt{KeyGen}(\lambda)$$
$$\texttt{pk}_{1-b} \in_\$ \mathcal{PK}(\lambda)$$

  That is, R generates one public key with a matching secret key, and another public key at random. R sends $\texttt{pk}_0, \texttt{pk}_1$ to the sender S.

- S encrypts each of its messages with one of the public keys:

$$c_0 \leftarrow \texttt{Enc}(\texttt{pk}_0, m_0)$$
$$c_1 \leftarrow \texttt{Enc}(\texttt{pk}_1, m_1)$$

  S sends c_0, c_1 to R.

- R decrypts the ciphertext c_b using its single secret key:

$$\texttt{Dec}(\texttt{sk}_b, c_b) = m_b$$

Security of this protocol is based on the security of the underlying public-key encryption scheme.

Let's now make this a bit more concrete. Our goal will be to reduce the problem of OT to a concrete computational assumption, and then to prove security of that scheme by constructing simulators.

Do to this, let's recall some background:

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:*

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

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.

Our OT protocol will be based on a type of public key encryption called *El-Gamal* encryption:

Definition 3 (El-Gamal Encryption)**.** *Parameterized by a family of groups $G = G_\lambda$ where DDH is assumed to be hard. Let g be the generator of G and q be the order G.*

- `KeyGen`*: Uniformly sample a number* $\texttt{sk} \in_\$ \mathbb{Z}_q$*. Then, set* $\texttt{pk} \leftarrow g^{\texttt{sk}}$*.*
- $\texttt{Enc}(\texttt{pk}, m)$*: Assume that $m \in G$ is a group element. Sample a random number $r \in_\$ \mathbb{Z}_q$. The ciphertext c is defined as follows:*

$$(g^r, \texttt{pk}^r \cdot m)$$

- $\texttt{Dec}(\texttt{sk}, c)$*: Let $c = (c_0, c_1)$. Compute the following:*

$$c_1 \cdot (c_0^{-\texttt{sk}})$$

Some notes about this encryption scheme:

- First, the scheme is correct. Namely, $\texttt{Dec}(\texttt{sk}, \texttt{Enc}(\texttt{pk}, m)) = m$:

$$\begin{aligned} c_1 \cdot (c_0^{-\texttt{sk}}) &= (\texttt{pk}^r \cdot m) \cdot (g^r)^{-\texttt{sk}} \\ &= ((g^{\texttt{sk}})^r \cdot m) \cdot (g^r)^{-\texttt{sk}} \\ &= ((g^{r \cdot \texttt{sk}}) \cdot m) \cdot (g^{-r \cdot \texttt{sk}}) = m \end{aligned}$$

- Second, the scheme is *secure*, in the sense that if DDH holds, ciphertexts look uniform to an adversary who does not know the secret key $\texttt{sk}$. In particular, for any message $m \in G$ and randomly sampled public key $\texttt{pk}$, the following ensembles are indistinguishable:

$$\{ \quad (\texttt{pk}, c_0, c_1) \quad \text{where} \quad (c_0, c_1) \leftarrow \texttt{Enc}(\texttt{pk}, m) \quad \} \overset{c}{=} \{ \quad (\texttt{pk}, g_0, g_1) \quad \text{where} \quad g_0, g_1 \in_\$ G \quad \}$$

  This is, in fact, quite immediate from DDH: c_0 is a uniform element g^r of G by construction, and $c_1 = (\texttt{pk}^r \cdot m) = g^{r \cdot \texttt{sk}} \cdot m$ uses $g^{r \cdot \texttt{sk}}$ as a one-time pad, which is of exactly the form required by the DDH assumption.

- Third, we sample public keys of this scheme uniformly at random, without calling `KeyGen`. In particular, an element g^r where $r \in_\$ \mathbb{Z}_q$ is *identically* distributed to an ElGamal public key.

Now, if we instantiate our above OT schematic with this particular encryption scheme, the result is indeed a semi-honest secure 1-out-of-2 OT scheme. We can prove this by constructing simulators.

First, we can show that this OT scheme has *perfect* security against a corrupted sender. In particular, the simulator for S is as follows. Recall that the simulator must simulate S's view, which includes (1) S's input, (2) S's received messages (the two random public keys), and (3) randomness (including randomness sampled when calling `Enc`. This can be done simply as follows:

```
$\texttt{Sim}_S(m_0, m_1)$ :
  $\texttt{pk}_0 \in_\$ \mathcal{PK}$
  $\texttt{pk}_1 \in_\$ \mathcal{PK}$
  $r_0 \in_\$ \mathbb{Z}_q$                randomness from $\texttt{Enc}(\texttt{pk}_0, m_0)$
  $r_1 \in_\$ \mathbb{Z}_q$                randomness from $\texttt{Enc}(\texttt{pk}_1, m_1)$
  return $(m_0, m_1, \texttt{pk}_0, \texttt{pk}_1, r_0, r_1)$
```

Again, this simulated view is *identically* distributed to the real-world view, because ElGamal public keys can be sampled uniformly at random.

Simulating the receiver R is a bit trickier, but still possible, based on the security property of ElGamal. In particular, R's simulator will not be *identically distributed* to R's real-world view, but merely indistinguishable from it.

The core of the simulation is that we must explain the two ciphertexts received by R, one corresponding to m_b and one corresponding to m_{1-b}. There are two tricks to realize for this simulation:

- We can simulate the ciphertext corresponding to m_b by "encrypting it ourselves".
- We can simulate the ciphertext corresponding to m_{1-b} by encrypting an arbitrary element of our choice, say the identity group element 1_G. This works because of the security of ElGamal and the fact that R does not hold a secret key for the ciphertext.

```
$\texttt{Sim}_R(b, m_b)$ :
  $\texttt{sk}_b \in_\$ \mathbb{Z}_q$
  $\texttt{pk}_b \leftarrow g^{\texttt{sk}_b}$
  $\texttt{pk}_{1-b} \in_\$ \mathcal{PK}$
  $c_b \leftarrow \texttt{Enc}(\texttt{pk}_b, m_b)$
  $c_{1-b} \leftarrow \texttt{Enc}(\texttt{pk}_b, 1_G)$
  return $(b, \texttt{sk}_b, \texttt{pk}_{1-b}, c_0, c_1)$
```

Semi-honest Security: Randomization and Multiple Parties

Our working definition for semi-honest security is as follows:

Definition 4 (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:*

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

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.

To understand this, let's define a randomized functionality and then (erroneously) apply Definition 4. We will see using Definition 4 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 function family* (PRF):

Definition 5 (Pseudorandom function family (PRF)). *Let $\mathcal{K}$ be a keyspace (indexed by a security parameter λ, let $\mathcal{D}$ be a domain, and let $\mathcal{R}$ be a range. Let $F : \mathcal{K} \times \mathcal{D} \to \mathcal{R}$ is considered a pseudorandom function family if the following two programs are indistinguishable:*

```
Real :
  private $k \in_\$ \{0,1\}^\lambda$
  public lookup($m$ : D) :
    return $F(k, m)$
```

$$\overset{c}{=}$$

```
Ideal :
  private dict ← empty-dictionary
  public lookup($m$ : D) :
    if $m \notin$ dict
      dict[$m$] $\in_\$$ R
    return dict[$m$]
```

Intuitively, a pseudorandom function $F(k, \cdot)$ appears to be a truly random function from the perspective of an adversary that does not know the key.

Now let's invent a possibly useful functionality:

Functionality 2 (Use-Once Oblivious PRF). *The functionality interacts with two parties A and B.*

- ***Parameters.*** *Parties agree on a PRF F.*
- ***Input.*** *A has no input. B has an input $x \in \mathcal{D}$.*
- ***Output.*** *A outputs a random PRF key $k \in_\$ \mathcal{K}$. B outputs $F(k, x)$.*

Note that this functionality is inherently *randomized*, because it outputs a random key to A. Thus, Definition 4 inherently cannot be used in the context of this functionality. 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 key $k \in_\$ \mathcal{K}$.
- A sends k to B and outputs k.

- B locally computes and outputs $F(k, x)$.

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

Exercise 1. *Construct simulators for A and B that "prove" this protocol secure in the context of Definition 4.*

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 6 (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 $\mathtt{Sim}_A$ (resp. $\mathtt{Sim}_B$)—called a simulator—such that for all inputs x, y, the following hold:*

$$\{\ \mathtt{View}_A^{\Pi(r)}(x,y), \mathtt{Output}^{\Pi(r)}(x,y)\ \} \overset{c}{=} \{\ (\mathtt{Sim}_A(x, z_0), (z_0, z_1)) \quad \textit{where} \quad (z_0, z_1) \leftarrow f(x,y)\ \}$$
$$\{\ \mathtt{View}_B^{\Pi(r)}(x,y), \mathtt{Output}^{\Pi(r)}(x,y)\ \} \overset{c}{=} \{\ (\mathtt{Sim}_B(y, z_1), (z_0, z_1)) \quad \textit{where} \quad (z_0, z_1) \leftarrow f(x,y)\ \}$$

Above, $\Pi(r)$ denotes to run Π with fixed randomness r s.t. A's view and the output are the result of the same protocol execution.

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 OPRF protocol.

Exercise 2. *Prove that for the above trivial OPRF protocol and for Definition 6, there is no simulator $\mathtt{Sim}_B$ for B.* Hint: You can use any such $\mathtt{Sim}_B$ to break the security property of the PRF F 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* subset of parties:

Definition 7 (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 subset of parties $C \subseteq \{0, ..., n-1\}$, there exists a probabilistic polynomial time algorithm $\mathtt{Sim}_C$—called a simulator—such that for all inputs $x_0, ..., x_{n-1}$, the following hold:*

$$\left\{\ \bigcup_{i \in C} \mathtt{View}_{P_i}^{\Pi(r)}(x_i, y_i), \mathtt{Output}^{\Pi(r)}(x_0, ..., x_{n-1})\ \right\}$$
$$\overset{c}{=} \left\{\ \mathtt{Sim}_C\left(\bigcup_{i \in C}(x_i, y_i)\right) \quad \textit{where} \quad (y_0, ..., y_{n-1}) \leftarrow f(x_0, ..., x_{n-1})\ \right\}$$

Above, $\Pi(r)$ denotes to run Π with fixed randomness r s.t. the corrupted subsets' view and the output are the result of the same protocol execution.

Exercise 3. *(The following is also a homework problem.) The above definition requires us to simulate all 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.*

Next Time

We will clean up our understanding of how to achieve MPC with/without OT. We will discuss protocol composition, show how to generalize our protocol to any number of parties, and discuss fundamental gaps between perfectly-secure and computationally-secure MPC protocols.