CMSC 858T—Secure Distributed Computation February 2, 2021

Lecture 3

Lecturer: Jonathan Katz *Scribe(s): Noemi Glaeser, Mackenzie Kong-Sivert*

1 Oblivious Transfer (OT)

Definition 1 Let $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ be a public-key encryption scheme. We say it has *oblivious key sampling* if there are efficient algorithms $\mathsf{SampKey}$, $\mathsf{SampRand}$ such that

$$\{pk \leftarrow \mathsf{Gen}(1^\kappa); r \leftarrow \mathsf{SampRand}(pk) : (r, pk)\}$$

and

$$\{r \leftarrow \{0,1\}^*; pk := \mathsf{SampKey}(1^\kappa; r) : (r, pk)\}$$

are computationally indistinguishable. ♢

The OT protocol works as follows:[1]

$\underline{P_1(x_0, x_1)}$		$\underline{P_2(b)}$
		$r_0, r_1 \leftarrow \{0,1\}^*$
		$(pk_b, sk_b) := \mathsf{Gen}(1^\kappa; r_b)$
		$pk_{1-b} := \mathsf{SampKey}(1^\kappa; r_{1-b})$
	P_2 sends pk_0, pk_1 to P_1	
$c_0 \leftarrow \mathsf{Enc}_{pk_0}(x_0)$		
$c_1 \leftarrow \mathsf{Enc}_{pk_1}(x_1)$	P_1 sends c_0, c_1 to P_2	$x_b := \mathsf{Dec}_{sk_b}(c_b)$

Theorem 1 *If the encryption scheme is CPA-secure and has oblivious key sampling,*[2] *then the above protocol securely realizes OT for semi-honest adversaries.*

Proof We present a proof for the case of a corrupted P_2. (The case of a corrupted P_1 is left as an exercise.) Define the following simulator $\mathcal{S}$: on input b and x_b (which $\mathcal{S}$ obtains by sending b to the trusted party implementing the OT functionality) do:

1. Choose $r_b \leftarrow \{0,1\}^*$ and compute $(pk_b, sk_b) := \mathsf{Gen}(1^\kappa; r_b)$.
2. Compute $(pk_{1-b}, sk_{1-b}) \leftarrow \mathsf{Gen}(1^\kappa)$ followed by $r_{1-b} \leftarrow \mathsf{SampRand}(pk_{1-b})$.
3. Compute $c_b \leftarrow \mathsf{Enc}_{pk_b}(x_b)$ and $c_{1-b} \leftarrow \mathsf{Enc}_{pk_{1-b}}(0)$.[3]
4. Output (r_0, r_1, c_0, c_1).

[1] We use "$\leftarrow$" for a randomized algorithm (including sampling uniformly from a set) and ":=" when we want to emphasize that a step involves a deterministic process.

[2] As a technical point, note that we require CPA-security and oblivious key sampling even against non-uniform adversaries. This is a consequence of out non-uniform definition of security computation.

[3] By '0' we mean a string of 0s of the correct length (which we assume is fixed as part of the protocol).

Fix some inputs x_0, x_1, b for the parties. The view of P_2 in the real world (executing the protocol above on these inputs) is distributed as

$$\mathsf{Real}(\kappa, x_0, x_1, b) = \left\{ \begin{array}{c} r_b, r_{1-b} \leftarrow \{0,1\}^*; \\ (pk_b, sk_b) \leftarrow \mathsf{Gen}(1^\kappa; r_b); \\ pk_{1-b} \leftarrow \mathsf{SampKey}(1^\kappa; r_{1-b}); \\ c_b \leftarrow \mathsf{Enc}_{pk_b}(x_b); c_{1-b} \leftarrow \mathsf{Enc}_{pk_{1-b}}(x_{1-b}) \end{array} : (r_0, r_1, c_0, c_1) \right\}.$$

The output of $\mathcal{S}$ in the ideal world is distributed as

$$\mathsf{Ideal}(\kappa, x_0, x_1, b) = \left\{ \begin{array}{c} r_b \leftarrow \{0,1\}^*; \\ (pk_b, sk_b) \leftarrow \mathsf{Gen}(1^\kappa; r_b); \\ (pk_{1-b}, sk_{1-b}) \leftarrow \mathsf{Gen}(1^\kappa); r_{1-b} \leftarrow \mathsf{SampRand}(pk_{1-b}); \\ c_b \leftarrow \mathsf{Enc}_{pk_b}(x_b); c_{1-b} \leftarrow \mathsf{Enc}_{pk_{1-b}}(0) \end{array} : (r_0, r_1, c_0, c_1) \right\}.$$

To show that the ensembles $\{\mathsf{Real}(\kappa, x_0, x_1, b)\}_{\kappa, x_0, x_1, b}$ and $\{\mathsf{Ideal}(\kappa, x_0, x_1, b)\}_{\kappa, x_0, x_1, b}$ are computationally indistinguishable, we consider a hybrid (i.e., intermediate) distribution:

$$\mathsf{Hybrid}(\kappa, x_0, x_1, b) = \left\{ \begin{array}{c} r_b \leftarrow \{0,1\}^*; \\ (pk_b, sk_b) \leftarrow \mathsf{Gen}(1^\kappa; r_b); \\ (pk_{1-b}, sk_{1-b}) \leftarrow \mathsf{Gen}(1^\kappa); r_{1-b} \leftarrow \mathsf{SampRand}(pk_{1-b}); \\ c_b \leftarrow \mathsf{Enc}_{pk_b}(x_b); c_{1-b} \leftarrow \mathsf{Enc}_{pk_{1-b}}(x_{1-b}) \end{array} : (r_0, r_1, c_0, c_1) \right\}.$$

Notice that in the hybrid distribution, the random tapes are generated as in the ideal world, while the ciphertexts are generated as in the real world.

We now show that Hybrid is indistinguishable from both Ideal and Real, which implies that Ideal and Real are indistinguishable from each other.

Claim 2 *If the encryption scheme is CPA-secure, then* Hybrid *and* Ideal *are indistinguishable.*

To see this, fix some x_0, x_1, b, and an efficient distinguisher $\mathcal{D}$. We construct the following adversary $\mathcal{D}'$ attacking CPA-security of the encryption scheme: On input a public key pk_{1-b} and a ciphertext c_{1-b} that is either an encryption (under pk_{1-b}) of x_{1-b} or 0, adversary $\mathcal{D}'$ does:

1. Sample $r_b \leftarrow \{0,1\}^*$, and compute $(pk_b, sk_b) \leftarrow \mathsf{Gen}(1^\kappa; r_b)$ followed by $c_b \leftarrow \mathsf{Enc}_{pk_b}(x_b)$.
2. Compute $r_{1-b} \leftarrow \mathsf{SampRand}(pk_{1-b})$.
3. Output $\mathcal{D}(r_0, r_1, c_0, c_1)$.

Note that if c_{1-b} is an encryption of 0, then the input to $\mathcal{D}$ is distributed exactly according to $\mathsf{Ideal}(\kappa, x_0, x_1, b)$, whereas if c_{1-b} is an encryption of x_{1-b}, then the input to $\mathcal{D}$ is distributed exactly according to $\mathsf{Hybrid}(\kappa, x_0, x_1, b)$. It follows from CPA-security of the encryption scheme that $\mathcal{D}$ cannot distinguish these distributions with non-negligible probability.

Claim 3 *If the encryption scheme has oblivious key sampling, then* Hybrid *and* Real *are indistinguishable.*

To see this, fix x_0, x_1, b, and an efficient distinguisher $\mathcal{D}$. We construct the following adversary $\mathcal{D}'$ breaking the oblivious key sampling property. On input (r_{1-b}, pk_{1-b}), adversary $\mathcal{D}'$ does:

1. Sample $r_b \leftarrow \{0,1\}^*$, and compute $(pk_b, sk_b) \leftarrow \mathsf{Gen}(1^\kappa; r_b)$ followed by $c_b \leftarrow \mathsf{Enc}_{pk_b}(x_b)$.

2. Compute $c_{1-b} \leftarrow \mathsf{Enc}_{pk_{1-b}}(x_{1-b})$.

3. Output $\mathcal{D}(r_0, r_1, c_0, c_1)$.

If (r_{1-b}, pk_{1-b}) are generated by computing $pk_{1-b} \leftarrow \mathsf{Gen}(1^\kappa)$ and $r \leftarrow \mathsf{SampRand}(pk)$, then the input given to $\mathcal{D}$ is distributed exactly according to $\mathsf{Hybrid}(\kappa, x_0, x_1, b)$. On the other hand, if (r_{1-b}, pk_{1-b}) are generated by choosing $r_{1-b} \leftarrow \{0,1\}^*$ and then setting $pk := \mathsf{SampKey}(1^\kappa; r)$, then the input given to $\mathcal{D}$ is distributed exactly according to $\mathsf{Real}(\kappa, x_0, x_1, b)$. It follows from the oblivious sampling property of the encryption scheme that $\mathcal{D}$ cannot distinguish these distributions with non-negligible probability.

This completes the proof. ■

It is worth noting that security of the protocol relies strongly on semi-honest behavior of the adversary. A malicious receiver could easily arrange to learn both inputs of the sender, and a malicious sender could learn something about the receiver's bit by computing one ciphertext correctly and the other incorrectly.

2 The GMW Protocol

We now show how to use OT to do secure multi-party computation of arbitrary functionalities.

First, we observe that it suffices to consider deterministic functions. To see this, we show how it is possible to compute an arbitrary randomized function g if parties have access to an ideal trusted party computing arbitrary deterministic functions. Given g, define the deterministic function

$$f((x_1, r_1), \ldots, (x_n, r_n)) = g(x_1, \ldots, x_n; r_1 \oplus \cdots \oplus r_n).$$

The protocol for securely computing g, when parties hold inputs $x_1, \ldots, x_n$, is:

1. Each party P_i chooses uniform r_i,

2. The parties invoke the ideal functionality computing f, with each party P_i using input (x_i, r_i).

One can verify that this is a perfectly secure protocol for computing g in the f-hybrid model, even with $n-1$ malicious corruptions.

We next show a protocol due to Goldreich, Micali, and Wigderson (the GMW protocol) for computing any deterministic function f given access to a trusted functionality computing OT. We start by expressing f as a boolean circuit consisting of NOT gates and AND/XOR gates of fan-in 2 and arbitrary fan-out. (Any function can be written using these gates.) We describe the protocol for the case of two parties, and leave the n-party case to next time.

The key idea is for the two parties to set up and maintain the following invariant: for every wire of the circuit, the parties hold a two-out-of-two sharing of the boolean value on that wire. (A two-out-of-two sharing of a bit b is a pair of bits (b_1, b_2) such that $b = b_1 \oplus b_2$.) Moreover, this sharing will be sufficiently random so that no information is leaked. The parties begin the protocol by setting up this invariant on the input wires, and then inductively maintain the invariant at each gate of the circuit. Thus, at a high level, the protocol consists of the three phases:

Input-sharing phase. Say P_1 holds the input bit a on some input wire of the circuit. P_1 chooses a uniform bit a_1, sets $a_2 := a \oplus a_1$, sends a_2 to P_2, and keeps a_1. The parties now have a sharing

(a_1, a_2) of the input bit a. The parties each do this for every input wire of the circuit for which they holds the corresponding input bit.

Note that after this step the parties have established the desired invariant on the input wires of the circuit.

Circuit-evaluation phase. We now show how the parties can inductively ensure the invariant at internal wires of the circuit. There are three types of gates to consider; the first two are easy but the last is more challenging:

- Consider a NOT gate $b = \neg a$, where the parties already have a sharing (a_1, a_2) of a. (We overload notation, and use a to refer both to the wire and the value on what wire as the circuit is evaluated.) Parties can generate a sharing of b by simply having one of the parties flip their share of a. E.g., if P_1 sets $b_1 := \neg a_1$ and P_2 sets $b_2 := a_2$ then (b_1, b_2) is a sharing of the correct value b.

- Consider an XOR gate $c = a \oplus b$, where the parties already have sharings (a_1, a_2) of a and (b_1, b_2) of b. The parties can generate a sharing of c by simply XORing their shares locally. I.e., P_1 sets $c_1 := a_1 \oplus b_1$ and P_2 sets $c_2 := a_2 \oplus b_2$. One can verify that (c_1, c_2) is a sharing of the correct value c.

- Finally, consider an AND gate $c = a \wedge b$, where the parties already have sharings (a_1, a_2) of a and (b_1, b_2) of b. There is no local computation the parties can do that will give them a sharing of c. (Can you prove this?) Instead, the parties proceed as follows:

  1. P_1 chooses a random bit c_1 that will be its share of c.
  2. Now P_1 has to determine what P_2's share of c should be. It can reason as follows: if $a_2 = 0$ and $b_2 = 0$, then $c = a_1 \wedge b_1$ and so $c_2 = (a_1 \wedge b_1) \oplus c_1$. Since P_1 knows a_1, b_1, this is an explicit value that P_1 can compute; call it c_2^{00}. Reasoning similarly, P_1 can compute values c_2^{01}, c_2^{10}, and c_2^{11}, where

     $$c_2^{a_2 b_2} = ((a_1 \oplus a_2) \wedge (b_1 \oplus b_2)) \oplus c_1.$$

     The problem is, P_1 does not know what values of a_2, b_2 are held by P_2! Instead, the parties will use 1-out-of-4 OT to make sure P_2 gets (only) the correct value. That is, P_1 will act as the sender with inputs $(c_2^{00}, c_2^{01}, c_2^{10}$, and c_2^{11}, and P_2 will act as the sender with input $a_2 b_2$ (i.e., a number in the range of 0 to 3, expressed in binary). Using an invocation of OT, P_2 will learn $c_2^{a_2 b_2}$ and take that as its share c_2 of c. The shares (c_1, c_2) held by the parties are a sharing of the correct value c by construction.

Output-reconstruction phase. The above allows the parties the ensure the invariant for the output wires of the circuit. Say the parties hold a sharing (a_1, a_2) of an output wire a. If a is an output wire that P_2 is supposed to learn at the end of the protocol, then P_1 can simply send a_1 to P_2, who can then compute $a := a_1 \oplus a_2$; the parties act analogously when P_1 is supposed to learn the output.

Correctness of the protocol follows by construction. As for security, we have:

Theorem 4 *The 2-party GMW protocol securely computes arbitrary functions f against semi-honest adversaries. In fact, it is perfectly secure in the OT-hybrid model.*

Proof We sketch a simulator $\mathcal{S}$ for the case when P_2 is corrupted. (The case when P_1 is corrupted is even easier.) $\mathcal{S}$ is given both the initial input of P_2 as well as the output that P_2 learns from evaluation of f. The simulator $\mathcal{S}$ works as follows:

- For each input wire a, choose random share a_2. (For input wires that are inputs of P_2, this will correspond to randomness uses by P_2; for input input wires that are inputs of P_1, this will correspond to the share sent by P_1 to P_2.)
- NOT gates and XOR gates involve only local computation, so there is nothing to simulate. For AND gates, $\mathcal{S}$ must only simulate the output of the OT functionality. This is done by simply setting the output from each invocation of the OT functionality to be a independent, uniform bit.
- For an output wire a whose value P_2 is supposed to learn, and which $\mathcal{S}$ knows (since it learned a from the ideal evaluation of f) let a_2 denote P_2's share of that wire. (Note that a_2 is defined by the view of P_2 simulated thus far.) Set $a_1 = a \oplus a_2$ and include a_1 in the view of P_2.

The above provides a *perfect* simulation of the view of P_2 in an execution of the GMW protocol in the OT-hybrid model. ■

By instantiating the ideal OT functionality with a secure OT protocol, we obtain a (computationally) secure real-world protocol.