CS 355 – Topics in Cryptography April 18, 2018

Lecture 6: Oblivious Transfer and Yao's Garbled Circuits

Instructors: Henry Corrigan-Gibbs, Sam Kim, David J. Wu

1 Introduction

In the last few lectures, we have been discussing proof systems. A proof system is an interactive protocol between two parties: a prover and a verifier. At the end of the protocol, the prover convinces the verifier whether some statement is true or not. In the next few lectures, we are going to consider a more general setting of *multiparty computation* (MPC). Today, we are going to restrict ourselves to just *two-party computation* (2PC).

Setting. A two party protocol is defined with respect to some joint function $f : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$ called the *functionality*. Let's refer to the two parties that are involved in the protocol as Alice and Bob.

- **Start of the protocol**: Alice holds some private input x, and Bob holds some private input y.
- **End of the protocol**: Both Alice and Bob learns the output $f(x, y)$.[1]

For security, we want each of the parties in the protocol to not learn any information about the other party's private input other than what can be inferred from the output $f(x, y)$.

Examples.

- *Yao's millionaire problem*: We have two millionaires Mark and Oprah. The two parties want to find out who is richer between the two. However, both parties do not want to reveal to each other how much money they actually have.
- *Online advertising*: Google and some Business want to find out how effective Google ads are.
  - Google's input: List of users who were shown Business's Ad.
  - Business's input: List of people who bought their product.

  At the end of the two party protocol, Google and Business can learn how many people actually bought Business's product via Google's ad.
- *Private contact discovery*: At the start of the protocol, a client has a list of contacts and a Signal server has a list of all users for the Signal app. At the end of the protocol, the client learns the set of Signal users in its contacts who use Signal while the Signal server learns nothing.
- *Zero-Knowledge*: At the start of the protocol, prover has input (x, w) and verifier has input x. At the end of the protocol, both parties learn $\mathcal{R}(x, w)$.

[1] More generally, we can consider the functionality to consist of two different functions f_1, f_2. We can require that at the end of the protocol, Alice learn $f_1(x, y)$ and Bob learns $f_2(x, y)$.

Security models. For 2PC (and MPC in general), there are two main models of security that we can consider.

- **Semi-honest**: Both Alice and Bob are guaranteed to follow the protocol specification exactly. At the end of the protocol, they look at the transcript of the protocol and try to extract more information about the other party's private input. This is analogous to the honest verifier zero-knowledge (HVZK) that we studied in the problem set 1.
- **Malicious**: Both Alice and Bob can deviate from the protocol specification at any time to fool the other party in providing more information about its own private input. This is analogous to the standard zero-knowledge (ZK) definition.

For this lecture, we will restrict ourselves to the *semi-honest* security model.

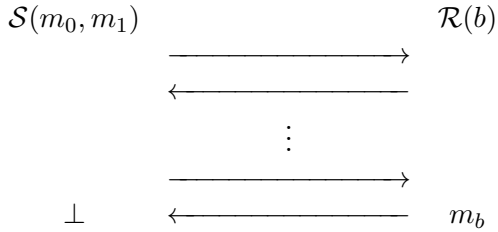
Semi-formal Definition. An interactive protocol $\langle A, B\rangle$ for a *functionality* $f : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$ must satisfy the following properties:

- **Correctness**: For all inputs $x, y \in \{0,1\}^*$,
$$\Pr[\mathsf{output}\,\langle A(x), B(y)\rangle = f(x,y)] = 1.$$
- **Privacy**: There exist efficient algorithms (simulators) Sim_A, Sim_B such that for all inputs $x, y \in \{0,1\}^*$,
$$\mathsf{Sim}_A(x, f(x,y)) \approx_c \mathsf{view}_A(\langle A(x), B(x)\rangle)$$
$$\mathsf{Sim}_B(y, f(x,y)) \approx_c \mathsf{view}_B(\langle A(x), B(x)\rangle)$$

2 Oblivious Transfer

There is a two-party computation functionality that is especially very useful called *oblivious transfer*. In an oblivious transfer (OT) protocol, there are two parties: a receiver $\mathcal{S}$ and a sender $\mathcal{R}$.

- **Start of the protocol**: At the start of the protocol, the sender $\mathcal{S}$ holds a pair of messages (m_0, m_1), and the receiver $\mathcal{R}$ holds some bit $b \in \{0,1\}$.
- **End of the protocol**: At the end of the protocol, the sender $\mathcal{S}$ learns nothing, while the receiver $\mathcal{R}$ learns m_b.



What are the security properties that we want from an OT protocol?

- **Sender Privacy**: At the end of the OT protocol, the receiver learns only m_b, and nothing else about m_{1-b}.

- **Receiver Privacy**: At the end of the OT protocol, the sender does not learn any information about the receiver's bit b.

Definition 2.1 (Oblivious Transfer)**.** A two party protocol $\langle \mathcal{S}, \mathcal{R} \rangle$ between a sender $\mathcal{S}$ and a receiver $\mathcal{R}$ is an *oblivious transfer* (OT) protocol if it satisfies the following properties:

- **Correctness**: For any input $m_0, m_1 \in \{0,1\}^*$,

$$\Pr[\mathsf{output}_{\mathcal{R}} \langle \mathcal{S}(m_0, m_1), \mathcal{R}(b) \rangle = m_b] = 1.$$

- **Privacy**:
  - *Sender Privacy*: There exists an efficient simulator $\mathsf{Sim}_{\mathcal{R}}$ such that for all pairs of messages (m_0, m_1) and $b \in \{0,1\}$:

$$\mathsf{view}_{\mathcal{R}}((m_0, m_1), b) \approx_c \mathsf{Sim}_{\mathcal{R}}(b, m_b).$$

  - *Receiver Privacy*: There exists an efficient simulator $\mathsf{Sim}_{\mathcal{S}}$ such that for all pairs of message (m_0, m_1) and $b \in \{0,1\}$:

$$\mathsf{view}_{\mathcal{S}}((m_0, m_1), b) \approx_c \mathsf{Sim}_{\mathcal{S}}(m_0, m_1).$$

2.1 Bellare-Micali Oblivious Transfer Construction [1]

Let $\mathbb{G}$ be a group of prime order p with generator g. Let H be a hash function $\mathbb{G} \to \{0,1\}^\ell$ (modeled as a random oracle). We let m_0 and m_1 be the sender's messages and let $b \in \{0,1\}$ be the receiver's input. Then, the protocol proceeds as follows:

1. The sender $\mathcal{S}$ chooses $c \xleftarrow{R} \mathbb{G}$ and sends c to the receiver $\mathcal{R}$.
2. The receiver $\mathcal{R}$ chooses a random key $k \xleftarrow{R} \mathbb{Z}_p$ and computes two ElGamal public keys $y_b \leftarrow g^k$ and $y_{1-b} \leftarrow \frac{c}{g^k}$, and sends y_0, y_1 to the sender.
3. If $y_0 \cdot y_1 \neq c$, then the sender $\mathcal{S}$ immediately aborts. Otherwise, $\mathcal{S}$ chooses $r_0, r_1 \leftarrow \mathbb{Z}_p$ and computes ElGamal ciphertexts $c_0 \leftarrow (g^{r_0}, H(y_0^{r_0}) \oplus m_0)$ and $c_1 \leftarrow (g^{r_1}, H(y_1^{r_1}) \oplus m_1)$. The sender sends c_0, c_1 to the receiver.
4. The receiver $\mathcal{R}$ parses the ciphertext $c_b = (v_0, v_1)$ and then decrypts using knowledge of k:

$$m_b = H(v_0^k) \oplus v_1.$$

   It outputs m_b.

Correctness of the scheme is by inspection. Let's reason about why this protocol is secure.

- **Sender Privacy.** Sender privacy follows from the CDH assumption.

  Intuition:

  1. The receiver $\mathcal{R}$ sends two ElGamal public keys y_0, y_1 to the sender $\mathcal{S}$, and $\mathcal{S}$ uses y_0, y_1 to encrypt each message m_0 and m_1. Therefore, as long as the receiver knows a corresponding decryption key for *one* of y_0, y_1 and *not both*, then $\mathcal{R}$ can only recover one of m_0 or m_1.

2. Since $c \in \mathbb{G}$ is generated by $\mathcal{S}$, the receiver $\mathcal{R}$ cannot generate decryption keys for both y_0, y_1 without solving the discrete log of c.

- **Receiver Privacy.** Receiver privacy follows information-theoretically.

  Intuition: The elements y_b and y_{1-b} are uniformly generated given that $y_b \cdot y_{1-b} = c$.

  (semi-)formal proof: We can construct a simulator Sim, which on input (m_0, m_1), works as follows:

  – Sample $c \xleftarrow{R} \mathbb{G}$ uniformly (as in the real protocol).
  – Sample $y_0, y_1 \leftarrow \mathbb{G}$ uniformly under the condition that $y_0 \cdot y_1 = c$.
  – Sample r_0, r_1, and compute $(g^{r_0}, H(y_0^{r_0}) \oplus m_0)$ and $(g^{r_1}, H(y_1^{r_1}) \oplus m_1)$ (as in the real protocol).

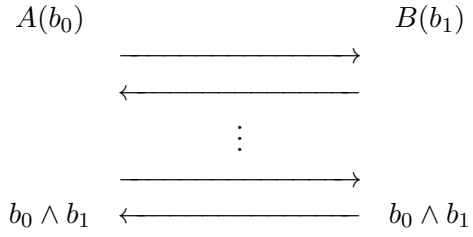
  Sim outputs the whole transcript $(c, y_0, y_1, (g^{r_0}, H(y_0^{r_0}) \oplus m_0), (g^{r_1}, H(y_1^{r_1}) \oplus m_1)$. It is easy to see that this is identically distributed as in the real protocol.

3 Garbled Circuits

A very useful tool for constructing two-party protocols is Yao's *garbled circuits* construction.[2] At a high level, *garbling* a circuits is a way of *encoding* a circuit C into "codes".

3.1 Warm-Up

For warm-up, let's consider a simple scenario where Alice holds a bit b_0 and Bob holds a bit b_1. They want to jointly compute the AND of their private bits $b_0 \wedge b_1$. How can they do this privately?



Let $E : \mathcal{K} \times \mathcal{M} \to \mathcal{C}$, $D : \mathcal{K} \times \mathcal{C} \to \mathcal{M}$ be an encryption scheme. Then, using *garbling*, let's construct the following protocol:

1. For each input variable to an AND gate, Alice samples a pair of keys (k_L^0, k_L^1), (k_R^0, k_R^1).

2. Then, Alice generates 4 ciphertexts according to the truth table of the AND gate.

α	β	AND
0	0	0
0	1	0
1	0	0
1	1	1

$\Longrightarrow$

α	β	AND
k_L^0	k_R^0	$c_{00} = E(k_L^0, E(k_R^0, 0))$
k_L^0	k_R^1	$c_{01} = E(k_L^0, E(k_R^1, 0))$
k_L^1	k_R^0	$c_{10} = E(k_L^1, E(k_R^0, 0))$
k_L^1	k_R^1	$c_{11} = E(k_L^1, E(k_R^1, 1))$

[2] Yao's garbled circuits is a very useful tool not only for two-party computation, but for all of cryptography in general.

3. Alice sends over the 4 ciphertexts $c_{00}, c_{01}, c_{10}, c_{11}$ in permuted order to Bob.
4. Alice also sends the corresponding key for its own input bit $k_L^{b_0}$ to Bob.
5. Alice and Bob proceeds in an oblivious transfer protocol where Alice plays the sender and Bob plays the receiver.
   - Alice's input: (k_R^0, k_R^1)
   - Bob's input: b_1
6. At the end of the OT protocol, Bob receives $k_R^{b_1}$.
7. Now Bob has 4 ciphertexts $c_{00}, c_{01}, c_{10}, c_{11}$ (in permuted order) and a pair of keys $k_L^{b_0}$, $k_R^{b_1}$. Bob tries to decrypt each ciphertext $D(k_R^{b_1}, D(k_L^{b_0}, c))$. Then, 3 out of the 4 ciphertexts should decrypt to some random garbage. 1 ciphertext should decrypt to either 0 or 1.
8. Bob sends back the bit he decrypted to Alice.

Correctness of the protocol is by inspection. Let's intuitively reason about why this protocol is actually secure by looking at each of Alice's view and Bob's view.

- Alice's view: The only messages that Alice actually receives from Bob throughout the protocol are (i) the OT protocol messages and (ii) the resulting bit $b_0 \wedge b_1$. The information $b_0 \wedge b_1$ is something that Alice is supposed to learn, so this is not a problem. Also, by security of the OT protocol, Alice does not learn any information about Bob's input b_1. Hence, for the whole protocol, Alice does not learn any more information about b_1 other than what can already be inferred from $b_0 \wedge b_1$.
- Bob's view: Let's consider all the messages that Bob receives from Alice
  - $c_{00}, c_{01}, c_{10}, c_{11}$: These are ciphertexts, which means that by semantic security, they don't provide any useful information to Bob as long as Bob is not provided the set of keys that can decrypt them.
  - $k_L^{b_0}$: This is a randomly generated key, and Bob does not know whether it corresponds to k_L^0 or k_L^1.
  - OT messages: By the security of OT, the only information that Bob learns is $k_R^{b_1}$.

  Hence, with access to only $k_L^{b_0}$, $k_R^{b_1}$, Bob only learns $b_0 \wedge b_1$ and nothing else.

3.2 General 2PC

Now let's extend the protocol from warm-up to general 2PC protocol for arbitrary functionality f. Instead of garbling a single gate, we are going to garble the whole circuit representation of f. How do we garble f?

- For each input wires and internal wires w of the circuit, assign a pair of keys (k_w^0, k_w^1).

- For each gate of the circuit, generate 4 ciphertexts which encrypts the corresponding key associated with the output wire according to the truth table of the gate.

α	β	AND
0	0	0
0	1	0
1	0	0
1	1	1

$\Longrightarrow$

α	β	AND
k_L^0	k_R^0	$c_{00} = E(k_L^0, E(k_R^0, k_{\mathsf{out}}^0))$
k_L^0	k_R^1	$c_{01} = E(k_L^0, E(k_R^1, k_{\mathsf{out}}^0))$
k_L^1	k_R^0	$c_{10} = E(k_L^1, E(k_R^0, k_{\mathsf{out}}^0))$
k_L^1	k_R^1	$c_{11} = E(k_L^1, E(k_R^1, k_{\mathsf{out}}^1))$

- For each gate connected to an output wire of the circuit, we encrypt 0/1 according to the truth table as before.
- **Invariant**: Given the 4 ciphertexts associated with a gate and a pair of keys corresponding to each input wire to the gate, one can compute a corresponding key for the output wire by decrypting each of the 4 ciphertexts (for the output wires, one decrypts 0/1).[3]

Using the above garbling method, we can construct a general two-party protocol. Without loss of generality, let's say $x, y \in \{0,1\}^n$ and $f : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^*$.

1. Alice represents the function f as a circuit and *garbles* f using the method above. f has a total of $2n$ input wires corresponding to $x_1, \ldots, x_n, y_1, \ldots, y_n$.
2. Alice sends over all of the ciphertexts that are generated for each gate to Bob (there are $4 \cdot |G|$ number of ciphertexts where $|G|$ represents the total number of gates in f).
3. Alice sends over the corresponding keys for its own input wires $k_1^{x_1}, \ldots, k_n^{x_n}$.
4. Alice and Bob proceeds in a total of n oblivious transfer protocol where Alice plays the sender and Bob plays the receiver. For $i = 1, \ldots, n$, the inputs to the ith OT is as follows:
   - Alice's input: (k_{n+i}^0, k_{n+i}^1)
   - Bob's input: y_i

   At the end of the protocol, Bob learns the keys $k_{n+1}^{y_1}, \ldots, k_{2n}^{y_n}$.
5. At this point, Bob knows a single key for each of the $2n$ input wires of the circuit. Bob evaluates the circuit using these keys (using the invariant condition above).
6. In the end, Bob learns the output $f(x, y)$.
7. Bob sends over $f(x, y)$ to Alice.

At the end of the protocol, both Alice and Bob learns the value $f(x, y)$.

Note on round complexity. The Bellare-Micali OT construction that we discussed in a 3-round protocol. There exists OT constructions that are 2-round (i.e. Naor-Pinkas [2]). Combining Yao's garbled circuit trick with OT in parallel, we can get a total of 3-round protocol for general 2PC (for semi-honest security model).

[3] Slight technicality: we must encode the output wire keys k_{out} such that a successful decryption is detectable. One easy way of doing this is just appending a bunch of zero string when encrypting. So you would actually encrypt $E(k_L^0, E(k_R^0, k_{\mathsf{out}}^0 | 00 \cdots 0))$ for example.

References

[1] Mihir Bellare and Silvio Micali. Non-interactive oblivious transfer and applications. In *Conference on the Theory and Application of Cryptology*, pages 547–557. Springer, 1989.

[2] Moni Naor and Benny Pinkas. Efficient oblivious transfer protocols. In *Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms*, pages 448–457. Society for Industrial and Applied Mathematics, 2001.