0368.4162 Foundations of Cryptography December 28, 2017

Lecture 10: Secure Computation against Semi-Honest Adversaries

Lecturer: Nir Bitansky *Scribes: Almog Elharar, Shahaf Nacson*

1 Previously on Foundations of Crypto

We stated the GMW theorem regarding the existence of secure computation protocols for arbitrary functions. We also discussed the GMW reduction from general adversaries to semi-honest adversaries that follow the protocol, and just try to gain information on honest parties.

2 Protocols against Semi-Honest Adversaries

Today we'll construct protocols against semi-honest adversaries. By now, there are several known ways of doing this, with different advantages. We'll start from the GMW construction. We will focus on the case of two parties, which already conveys the main underlying ideas (we will hint how to generalize these ideas in the homework).

A Warmup: Computing Functions with Short Inputs using Oblivious Transfer. Consider the restricted case where Alice and Bob have short private inputs $a, b \in \{0,1\}^k$, where $k = O(\log n)$. They would like to compute a deterministic function $f(a,b) \in \{\bot\} \times \{0,1\}$, where only Bob gets an output. Designing a semi-honest protocol for this task means that the protocol is correct and satisfies the following privacy requirement:

- There exists a PPT simulator SIM_A such that

$$\{VIEW_A(A(a), B(b))\}_{a,b} \approx_c \{SIM_A(a)\}_{a,b} \ ,^1$$

  where $(A(a), B(b))$ denotes an interaction between A and B with corresponding inputs a, b, and $VIEW_A$ includes a, the randomness of A, and all messages received from b in an interaction.

- There exists a PPT simulator SIM_B such that

$$\{VIEW_B(A(a), B(b))\}_{a,b} \approx_c \{SIM_B(b, f(a,b))\}_{a,b} \ .$$

We will achieve this using a tool called *oblivious transfer* (OT). In a $(1,t)$-OT protocol, a sender S holds t bits $\sigma_1, \ldots, \sigma_t$ and a receiver R holds a choice $i \in [t]$. The goal is for the receiver to learn σ_i, but gain no information on any σ_j, for $j \neq i$. The sender should not learn anything about the receiver choice. Formally, it is just a secure computation protocol for the function

$$((\sigma_1, \ldots, \sigma_t), i) \longmapsto (\bot, \sigma_i) \ .$$

Let's spell out the security definition for the case of semi-honest adversaries.

Definition 2.1 (Semi-Honest Security for OT)**.** *A $(1,t)$-OT protocol (S,R) is secure against semi-honest corruptions if*

- **Receiver Privacy:** *There exists a PPT simulator SIM_S such that*

$$\{VIEW_S(S(\sigma_1, \ldots, \sigma_t), R(i))\}_{\vec{\sigma},i} \approx_c \{SIM_S(\sigma_1, \ldots, \sigma_t)\}_{\vec{\sigma},i} \ .$$

[1] Formally, this indistinguishability has to hold even given the honest party's output $f(a,b)$. Since f is deterministic and a and b are fixed (and not changed by the adversary), the above definition is equivalent.

- **Sender Privacy:** *There exists a PPT simulator SIM_R such that*

$$\{VIEW_R(S(\sigma_1,\ldots,\sigma_t),R(i))\}_{\vec{\sigma},i} \approx_c \{SIM_R(i,\sigma_i)\}_{\vec{\sigma},i} \ .$$

We can immediately see the connection between OT and secure computation with (even just one out of two) short inputs.

Claim 2.2. *Let $k = O(\log n)$, then $(1,2^k)$-OT implies secure computation for any (efficient) function*

$$f : \{0,1\}^n \times \{0,1\}^k \to \{\bot\} \times \{0,1\} \ .$$

Proof Sketch. A transforms its input $a \in \{0,1\}^n$ into the truth table of the function $f(a,\cdot)$:

$$(\sigma_1,\ldots,\sigma_{2^k} \mid \sigma_j = f(a,j)) \ ,$$

where j is interpreted as a string in $\{0,1\}^k$. B's input $b \in \{0,1\}^k$ is interpreted as an integer $b \in [2^k]$.

They run the OT protocol with the above inputs. The correctness and privacy requirements follow readily (make sure you see why). □

An OT protocol from Trapdoor Permutations [EGL82]. We now show a simple construction of a semi-honest $(1,t)$-OT protocol. Let (G,F,F^{-1}) be a TDP mapping $\{0,1\}^n$ to $\{0,1\}^n$ with a (deterministic) hardcore bit $B : \{0,1\}^n \to \{0,1\}$. Consider the following protocol:

$(S(\sigma_1,\ldots,\sigma_t),R(i))$:

- S samples keys $(sk,pk) \leftarrow G(1^n)$, and sends pk to R.
- R samples at random $y_1,\ldots,y_{i-1},y_{i+1},\ldots,y_t \leftarrow \{0,1\}^n$. The missing image y_i is sampled together with a preimage, i.e. $y_i = F_{pk}(x_i)$ for $x_i \leftarrow \{0,1\}^n$. All images $y_1,\ldots,y_t$ are sent to S.
- S then uses the trapdoor sk to recover $x_1,\ldots,x_t$ where $x_j = F_{sk}^{-1}(y_j)$. It then sends R

$$B(x_1)\oplus\sigma_1,\ldots,B(x_t)\oplus\sigma_t \ .$$

- R computes $B(x_i)$ and recovers σ_i.

Claim 2.3. *The protocol is correct and private.*

Proof Sketch. Correctness follows readily. We focus on privacy.

We show how to simulate each of the parties:

- $SIM_S(\sigma_1,\ldots,\sigma_t)$: simulates R's message $y_1,\ldots,y_t$ by simply sampling them at random from $\{0,1\}^n$. The rest of the view is sampled honestly. Here the simulation is perfect.
- $SIM_R(i,\sigma_i)$: samples (sk,pk), throws away sk, and uses pk to simulate the first sender message. It then samples $y_1,\ldots,y_t$ just like the honest receiver. To simulate the last sender message $c_1,\ldots,c_t$ it samples $c_1,\ldots,c_{i-1},c_{i+1},\ldots,c_t$ uniformly at random, and sets $c_i = B(x_i)\oplus\sigma_i$. Here the validity of simulation us shown based on the pseudorandomness of

$$B(F^{-1}(y_1)),\ldots,B(F^{-1}(y_{i-1})),B(F^{-1}(y_{i+1})),\ldots,B(F^{-1}(y_t)) \ .$$

□

Remark: Note that the above protocol is completely insecure in the presence of malicious parties. In fact, the protocol is insecure even for adversaries that behave completely honestly, except for choosing their randomness maliciously, which demonstrates the necessity of the coin-tossing-into-the-well step that we saw last lecture as part of the GMW compiler.

Dealing with Arbitrary Functions. We would now like to show how to construct semi-honest protocols for arbitrary (efficient) functions $f : \{0,1\}^n \times \{0,1\}^n \to \perp \times \{0,1\}$ (note that this is sufficient for computing functions where both parties get the output, as well as functions with long outputs).

The basic idea is that any efficient function $f(a,b)$, where now $a, b \in \{0,1\}^n$ can be represented as a composition of functions with small inputs. Specifically, we can represent it as a boolean circuit over some universal set of gates, e.g. $\{\oplus, \wedge\}$. From hereon we will directly think of f as such a boolean circuit.

We would now like to reduce the secure computation of f to securely computing each one of its gates. We already have a protocol for computing any specific gate. However, we cannot naïvely use this protocol to compute f. Why? If we did, the parties would in particular learn intermediate values the computation of f, which they're not supposed to learn.

Solution: Computing Over Secret Shares. Roughly speaking, the idea behind the solution is to split the actual computation, consisting of all intermediate wire values, $v_1, \ldots, v_{|f|}$ into two shares $x_1, \ldots, x_{|f|}$ and $y_1, \ldots, y_{|f|}$ so that each of the shares individually leaks no information (it is completely random), whereas any two shares x_i and y_i together allow to recover v_i. Our goal will be to augment the computation so that A only learns the x shares and B only learns the y shares. Eventually, they will only exchange the shares for the output of the function, which they're supposed to learn.

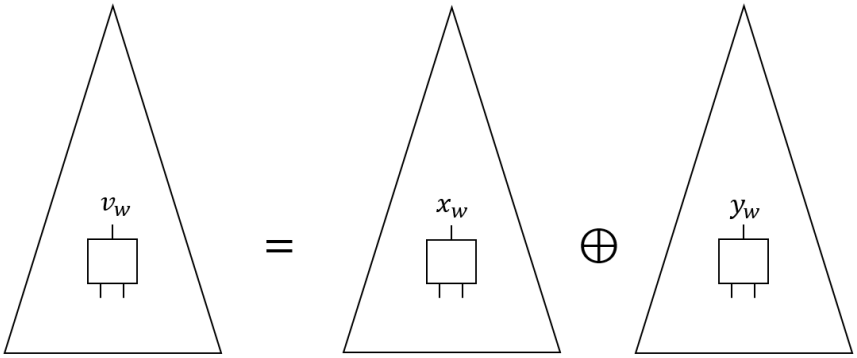

Figure 1: The original circuit computation is shared between A and B. The value v_u at each wire w is split into random shares x_w, y_w which are held by A, B respectively, and $v_w = x_w \oplus y_w$.

Specifically, the secret shares for any bit value v will be two random bits x and y subject to $v = x \oplus y$. We will augment every gate $g(a,b)$ into a new randomized function $\hat{g}((x_a, x_b), (y_a, y_b)) = (x, y)$,[2] which given two inputs representing shares of a and b, outputs two new shares x and y, which are random subject to $x \oplus y = g(x_a \oplus y_a, x_b \oplus y_b)$, which if everything was done correctly would be equal to $g(a,b)$.

Protocol (A,B) for computing a circuit f:

1. **Input Sharing:** Let $a \in \{0,1\}^n$ and $b \in \{0,1\}^n$ be the inputs.
   - A samples at random $x_a \leftarrow \{0,1\}^n$, stores it, and sends $y_a = a \oplus x_a$ to B.
   - B samples at random $y_b \leftarrow \{0,1\}^n$, stores it, and sends $x_b = b \oplus y_b$ to A.
2. **Gate by Gate Computation:** We assume w.l.o.g that the circuit computing f is layered, meaning that the input wires for gates at layer ℓ are the output wires of gates in layer $\ell - 1$. The computation proceed layer by layer. Initially the two parties A and B hold shares x_w and y_w, respectively, for each input wire w. For each ℓ, having computed shares for all input wires at layer ℓ, they compute shares for the layer's output wires. Specifically, for each gate $g(\cdot,\cdot)$ with input wires α, β and output wire γ:

[2] We will actually implement $\hat{g}$ through the computation of a deterministic function.

- A holds $x_\alpha, x_\beta \in \{0,1\}$ and B holds $y_\alpha, y_\beta \in \{0,1\}$.
- To obtain shares x_γ and y_γ, the parties do the following:
  - A samples $x_\gamma \leftarrow \{0,1\}$.
  - The two parties compute the function

$$\hat{g}((x_\alpha, x_\beta, x_\gamma), (y_\alpha, y_\beta)) = (\perp, y_\gamma) \quad \text{where} \quad y_\gamma = x_\gamma \oplus g(x_\alpha \oplus y_\alpha, x_\beta \oplus y_\beta)$$

    using the previously described protocol for functions with short input (indeed here B's input consists only of two bits).

3. **Output Reconstruction:** At the end of the protocol A and B hold shares x_o and y_o for the output wire o. A sends x_o to B who recovers the output $x_o \oplus y_o$.

Claim 2.4. *The protocol is correct and private.*

Proof Sketch. Correctness follows by construction.

We show how to simulate each of the parties:

- $SIM_A(a)$: Samples random values x_w for every wire w. For each gate g with input wires i, j and output wire k, run the OT sender simulator $SIM_S(x_\alpha, x_\beta, x_\gamma)$, to simulate the messages in the $\hat{g}$ protocols.
- $SIM_B(b, f(a,b))$: Samples random values y_w for every wire w. For the output wire o, it samples $x_o = y_o \oplus f(a,b)$. For each gate g with input wires i, j and output wire k, run the OT receiver simulator $SIM_R(y_\alpha, y_\beta)$ to simulate the messages in the $\hat{g}$ protocols.

To see that the simulation is valid, first note that putting aside the messages in the $\hat{g}$ protocols, the view of either party in the real protocol is identically distributed as it simulated view. Thus, to show that indistinguishability of the full views holds, it is enough to show indistinguishability for any fixing of all inputs for the $\hat{g}$ protocols. This is shown by a hybrid argument, where we gradually change every real $\hat{g}$ messages, to simulated messages. (It's worth taking the time to yourself to spell out this argument.) □

Remarks:

1. **The Original GMW Construction.** The GMW construction is based on the above ideas, but is actually described slightly differently. Specifically we consider the specific universal set of gates $\{+, \times \mod 2\}$. One benefit of this representation is that we only need to run a protocol for each multiplication gate $\times$, and not for $+$ gates (try to think why). Another benefit is that this representation admits a rather natural generalization to the m-party case (we will explore this in the homework).
2. **Composition.** In the above protocol, we encountered a common phenomena in general protocol design — to construct a protocol for the function f, we used a subroutine a protocol for a function $\hat{g}$. A useful approach that could greatly simplify both the design and analysis is to prove general *composition theorems*, which roughly say — *if you can construct a protocol π^g for f using an ideal trusted party for g, and you can construct a protocol τ for g, then you get a secure protocol π^τ for f.* Indeed, such composition theorems can be proved, but exhibit various subtleties that mostly have to do with how exactly the protocols are composed (in particular, there turns out to be an essential difference between composing protocols sequentially and composing protocols concurrently).

3 Garbled Circuits

One caveat of the GMW protocol is that it involves many rounds of interaction — as we described it, at least as the number of gates in f, but it can in fact be reduced to the depth of the circuit f, by handling in parallel all gates in a single layer. We will now discuss an alternative approach to two-party computation known as *Yao's garbled circuit*. This approach will give rise to constant-round secure computation, and will in fact be useful beyond the setting of secure computation.

Informally, a garbling scheme takes a circuit and an input (f, x) and encodes them into $(\hat{f}, \hat{x})$ so that:

1. $(\hat{f}, \hat{x})$ can be used to compute $f(x)$, but don't reveal any other information on x.
2. Computing $(f, x) \mapsto (\hat{f}, \hat{x})$ is in some sense simpler than computing $(f, x) \mapsto f(x)$.

There are several ways of formalizing the above two requirements and we will choose a specific one sufficient for our purposes.

Definition 3.1 (Garbling Scheme)**.** *A garbling scheme consists of three algorithms* $(CEnc, IEnc, Eval)$ *satisfying the following:*

- **Syntax:**
  - $CEnc_{sk}(f)$ *takes a secret key* sk *and a circuit* f*, and outputs an encoding* $\hat{f}$*.*
  - $IEnc_{sk}(x)$ *takes a secret key* sk *and an input* x*, and outputs an encoding* $\hat{x}$*.*
  - $Eval(\hat{f}, \hat{x})$ *takes as input encodings* $\hat{f}$ *and* $\hat{x}$ *and outputs a value* y*.*
- **Correctness:** *for any* f, x*, and* sk*,*

$$\Pr\left[Eval(\hat{f}, \hat{x}) = f(x) \;\middle|\; \begin{array}{c} \hat{f} \leftarrow CEnc_{sk}(f) \\ \hat{x} \leftarrow IEnc_{sk}(x) \end{array} \right] = 1$$

- **(Input) Privacy:** *there exists a PPT simulator SIM such that*

$$\left\{ \hat{f}, \hat{x} \;\middle|\; \begin{array}{c} k \leftarrow \{0,1\}^n \\ \hat{f} \leftarrow CEnc_{sk}(f) \\ \hat{x} \leftarrow IEnc_{sk}(x) \end{array} \right\}_{f,x} \approx_c \{SIM(f, f(x)\}_{f,x} \ .$$

- **ℓ-Locality:** *any bit of* $\hat{x}$ *depends on a (fixed) size-*ℓ *subset* S *of the bits of* x*. (typically, we'll be interested in small* ℓ*, say constant or logarithmic).*

Remark: the above definition may come in different colors and flavors, mostly with respect to what it means for the encoding to be simple. Above we only asked that the input encoding is local in the input x. A somewhat more common requirement is that the entire encoding process (function and input together) is local in the input and key. (See further details in [App11].)

Semi-Honest Computation from Garbled Circuits. Before we see how to construct them, let's see how to construct constant-round semi-honest two-party protocols.

Claim 3.2. *Let* $\ell = O(\log n)$*. Assuming a garbling scheme with locality* ℓ *and a* $(1, 2^\ell)$*-OT, there exists a constant-round two-party semi-honestly-secure protocol.*

Proof Sketch. We'll sketch the protocol.

Protocol (A, B) **for Computing** f**:**

1. A samples a secret key $sk \leftarrow \{0,1\}$. It then encodes the circuit $\hat{f} \leftarrow CEnc_{sk}(f)$, and sends $\hat{f}$ to B.
2. The parties jointly compute an encoding $\hat{x}$ of their inputs $x = (a, b)$. Specifically, for each bit $\hat{x}_\alpha$ they run an OT-based protocol where A inputs (k, a) and Bob inputs the ℓ-size subset of relevant bits. The protocols are ran in parallel.

Simulating the view of A is done by running the sender simulator for the OT-based protocols. Simulating the view of B is done by first running the simulator $SIM(f(a, b), f)$ for the garbling scheme to obtain simulated $\hat{x}, \hat{f}$, and then using the receiver simulator with the corresponding outputs take from $\hat{x}$. □

Yao's Garbled Circuit. The first garbling scheme was proposed by Yao.

Theorem 3.3 ([Yao86])**.** *Assuming OWFs, there exists a circuit garbling scheme with locality* 1.

Proof. We shall describe the scheme. We'll assume for simplicity that we have a secret-key encryption scheme (E, D) such that for any two keys $k \neq k'$ and any message m, it holds that $D_{k'}(E_k(m)) = \perp$. We'll explain later how to remove this assumption.

- For every wire w in the circuit, sample two secret keys k_w^0, k_w^1. Every secret key for the garbling scheme consists of all such wire keys $sk = \left\{k_w^0, k_w^1\right\}_w$.
- For every gate g with input wires α, β and output wire γ, we create a table T_g of encryptions:

$E_{k_\alpha^0}(E_{k_\beta^0}(k_\gamma^{g(0,0)}))$
$E_{k_\alpha^0}(E_{k_\beta^1}(k_\gamma^{g(0,1)}))$
$E_{k_\alpha^1}(E_{k_\beta^0}(k_\gamma^{g(1,0)}))$
$E_{k_\alpha^1}(E_{k_\beta^1}(k_\gamma^{g(1,1)}))$

  we then permute the rows of T_g at random and denote the permuted table by $\widetilde{T}_g$. The encoding $\hat{f}$ consists of all permuted tables $\left\{\widetilde{T}_g\right\}_g$ as well as a mapping $\left(b \mapsto k_o^b\right)_b$ for the output wire o.
- An encoding $\hat{x}$ of an input x consists of the keys $\{k_i^{x_i}\}$ corresponding to the input wires $1, \ldots, |x|$.
- To evaluate $\hat{f}, \hat{x}$, we can learn for each wire w exactly one key $k_w^{v(w)}$ corresponding to the value $v(w)$ of this wire in the computation. This is done all the way to the output, where we can map the final key to the output bit.

Simulation. To show that the privacy requirement holds, we'll describe the corresponding simulator $SIM(f, f(x))$:

- For every wire w in the circuit, SIM samples two secret keys k_w^0, k_w^1.
- For every gate g with input wires α, β and output wire γ, in contrast to the original table, SIM creates a table T_g where a single output key is encrypted in all entries, regardless of the gate:

$E_{k_\alpha^0}(E_{k_\beta^0}(k_\gamma^0)$
$E_{k_\alpha^0}(E_{k_\beta^1}(k_\gamma^0))$
$E_{k_\alpha^1}(E_{k_\beta^0}(k_\gamma^0))$
$E_{k_\alpha^1}(E_{k_\beta^1}(k_\gamma^0))$

  One exception is the output gate 0 where SIM encrypts in every tow the key $k_0^{f(x)}$ corresponding to the right output. SIM then permutes the rows of T_g as before.
- An encoding $\hat{x}$ of an input x consists of the keys $\left\{k_i^0\right\}$ corresponding to the input wires $1, \ldots, |x|$.
- The encoding $\hat{f}$ consists of all permuted tables $\left\{\widetilde{T}_g\right\}_g$ and the mapping $\left(b \mapsto k_o^b\right)_b$.

The validity of the simulator is proven by a hybrid argument. Let d be the number of layers in the circuit. For for $i \in \{0, \ldots, d\}$, in Hybrid H_i, for any gate g in layer i with output wire w, the table of encryptions encrypts in each one of the four rows the secret key $k_w^{v(w)}$ corresponding to the true value $v(w)$ of the computation $f(x)$ on the wire w. All garbled tables in layers $0, \ldots, i-1$ are simulated — they always encrypt the zero

key. All tables in layers $i+1, \ldots, n$ are generated as in a real garbled circuit — they encrypt different keys in each row corresponding to the computation of a gate. Note that H_0 corresponds to a real garbling and H_d corresponds to a simulated garbling.

To move from H_i to H_{i+1} we consider an intermediate hybrid $H_{i,i+1}$ where the garbled tables in layer $i+1$ are changed to be as in hybrid H_{i+1} — namely all four encryptions encode the key corresponding to the true value $v(w)$, instead of different keys corresponding to the computation.

The indistinguishability between H_i and H_{i+1} follows by the security of encryption. Crucially in H_i, the encrypted keys to be used in the next layer all correspond to the true value $v(w)$, and the key $k_w^{1-v(w)}$ is not in the adversary's view. Thus we can use the security of these keys to change the garbled tables in layer $i+1$. The indistinguishability of $H_{i,i+1}$ and H_{i+1} is completely information theoretic (it is basically a change of names).

□

References

[App11] Benny Applebaum. Randomly encoding functions: A new cryptographic paradigm - (invited talk). In *Information Theoretic Security - 5th International Conference, ICITS 2011, Amsterdam, The Netherlands, May 21-24, 2011. Proceedings*, pages 25–31, 2011.

[EGL82] Shimon Even, Oded Goldreich, and Abraham Lempel. A randomized protocol for signing contracts. In *Advances in Cryptology: Proceedings of CRYPTO '82, Santa Barbara, California, USA, August 23-25, 1982.*, pages 205–210, 1982.

[Yao86] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *27th Annual Symposium on Foundations of Computer Science, Toronto, Canada, 27-29 October 1986*, pages 162–167, 1986.