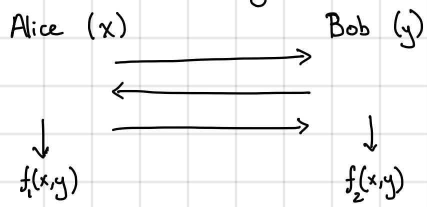
CS 6501 Week 9: Multiparty Computation

Interactive proofs are two-party protocols between a prover and a verifier, where prover's goal is to convince verifier that some statement x is true. This week, we consider a generalization to two-party computation:



[Alice has a (secret) input x and Bob has (secret) input y and they want to jointly compute $f(x,y)$ without revealing their inputs x, y to each other]

Examples: Yao's millionaire problem: Alice and Bob are millionaires and they want to learn which one of them is richer without revealing to the other their net worth [in this case $f(x,y) = 1$ if $x > y$ and 0 otherwise]

Private contact discovery: Client has a list of contacts on their phone while Signal (private messaging application) has list of users that use the service. Client wants to learn list of Signal users that are in their contact list while Signal server should learn nothing.

Private ML: Client has a feature vector x while the server has a model M. At the end, client should learn $M(x)$ and server should learn nothing.

Genome Privacy: Two patients want to identify if they share any rare genomic variants but do not wish to reveal their full genomes to one another.

Zero Knowledge: Prover has input (x, w) and verifier has input x. At the end of the protocol, verifier learns $R(x,w)$ while prover learns nothing.

Let $f = (\underbrace{f_1}_{\text{Party 1's output}}, \underbrace{f_2}_{\text{Party 2's output}})$ be a two-party functionality, and let π be an interactive protocol for computing f.

↳ We write $\text{view}_i^\pi(x,y)$ to denote the view of party $i \in \{1,2\}$ on a protocol invocation π on inputs x and y. Note that $\text{view}_i^\pi(x,y)$ is a random variable containing Party i's input, randomness, and all of the messages Party i received during the protocol execution.

↳ We write $\text{output}^\pi(x,y)$ to denote the output of protocol π on inputs x and y. We will write $\text{Output}^\pi(x,y) = (\text{output}_1^\pi(x,y), \text{output}_2^\pi(x,y))$ to refer to the outputs of the respective parties. The value $\text{output}_i^\pi(x,y)$ can be computed from $\text{view}_i^\pi(x,y)$.

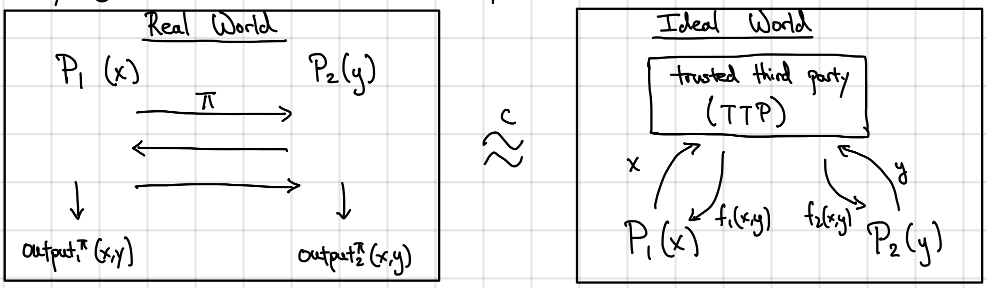
The protocol π should satisfy the following properties:

- Correctness: For all inputs x, y:
$$\Pr[\text{output}_i^\pi(x,y) = f_i(x,y)] = 1.$$
- (Semi-Honest) Security: There exist efficient simulators S_1 and S_2 such that for all inputs x and y
$$\{S_1(1^\lambda, x, f_1(x,y)), f(x,y)\} \stackrel{c}{\approx} \{\text{view}_1^\pi(x,y), \text{output}^\pi(x,y)\}$$
$$\{S_2(1^\lambda, y, f_2(x,y)), f(x,y)\} \stackrel{c}{\approx} \{\text{view}_2^\pi(x,y), \text{output}^\pi(x,y)\}$$

Notes:
- Security definition says that the view of each party can be simulated just given the party's input and its output in the computation (i.e., the minimal information that needs to be revealed for correctness). In other words, no additional information revealed about other party's input other than what is revealed by the output of the computation.
- Definition does not say other party's input is hidden. Only true if f does not leak the other party's input.
- Definition only requires simulating the view of the honest party. Thus, security only holds against a party that is "semi-honest" or "honest-but-curious": party follows the protocol as described, but may try to infer additional information about other party's input based on messages it receives.

Oftentimes, semi-honest security not good enough. Real adversaries can be <u>malicious</u> (i.e., deviate arbitrarily from protocol to corrupt the computation (e.g., cause honest users to compute the wrong answer, or worse, learn information about honest party's secret inputs)

Defining security against malicious adversaries is not easy. Here is a sketch (informal) of how it is typically done:



<u>Security</u>: An adversary that corrupts P_i in the real world can be simulated by an ideal adversary that corrupts P_i in the ideal world. Output of real and ideal executions consists of the adversary's output and the outputs of the honest parties. Ideal execution designed to capture world where no attacks are possible. Only possible adversarial behavior is "lying" about input to the execution (output is computed by the honest parties).

<u>Fairness</u>: Adversary should not be able to learn outputs of the computation before the honest parties

[Imagine a secure auction where adversary learns results first and decides to abort the protocol and claim "network failure" before honest parties can obtain the results]

- Difficult notion to achieve (beyond the scope of this course)

<u>Our focus</u>: Semi-honest two-party computation

this is necessary and sufficient for <u>general</u> multiparty computation (MPC)!

<u>Key cryptographic building block</u>: oblivious transfer (OT)

sender has two messages m_0, m_1

receiver has a bit $b \in \{0,1\}$

at the end of the protocol, receiver learns m_b, sender learns nothing

<u>Correctness</u>: For all messages $m_0, m_1 \in \{0,1\}^n$:

$$\Pr[\text{output}^{OT}((m_0, m_1), b) = (\perp, m_b)] = 1$$

<u>Sender Security</u>: There exists an efficient simulator S such that for all $m_0, m_1 \in \{0,1\}^n$, $b \in \{0,1\}$

$$S(1^\lambda, b, m_b) \overset{c}{\approx} \text{view}_2((m_0, m_1), b)$$

Receiver's view can be simulated just given choice bit b and chosen message m_b (message m_{1-b} remains hidden).

<u>Receiver Security</u>: There exists an efficient simulator S such that for all $m_0, m_1 \in \{0,1\}^n$ and $b \in \{0,1\}$,

$$S(1^\lambda, m_0, m_1) \overset{c}{\approx} \text{view}_1((m_0, m_1), b)$$

Sender's view can be simulated just given its input messages m_0, m_1 (receiver's choice bit b is hidden).

Constructing oblivious transfer: Very heavily-studied primitive and protocols. We will look at two examples.

Bellare - Micali OT: Let $\mathbb{G}$ be a prime order group and $H: \mathbb{G} \to \{0,1\}^n$ be a hash function (modeled as a random oracle):

sender $(m_0, m_1 \in \{0,1\}^n)$ — receiver $(b \in \{0,1\})$

Sender: $c \xleftarrow{R} \mathbb{G}$

$\xrightarrow{\quad c \quad}$

Receiver: $s_b \xleftarrow{R} \mathbb{Z}_p$, $h_b \leftarrow g^{s_b}$, $h_{1-b} \leftarrow c / g^{s_b}$ — view as ElGamal public keys (s_b is secret key for h_b)

$\xleftarrow{\quad h_0, h_1 \quad}$

Sender:

$r_0 \xleftarrow{R} \mathbb{Z}_p \quad ct_0 \leftarrow (g^{r_0}, H(h_0^{r_0}) \oplus m_0)$

$r_1 \xleftarrow{R} \mathbb{Z}_p \quad ct_1 \leftarrow (g^{r_1}, H(h_1^{r_1}) \oplus m_1)$

$\xrightarrow{\quad ct_0, ct_1 \quad}$

Receiver: $m_b \leftarrow H(ct_{b,0}^{s_b}) \oplus ct_{b,1}$

Correctness: By construction, $ct_{b,0}^{s_b} = (g^{r_b})^{s_b} = h_b^{r_b}$ and correctness follows.

Sender Security: We construct simulator as follows. On input $(1^\lambda, b, m_b)$:

1. Choose $c \xleftarrow{R} \mathbb{G}$
2. Choose $s_b \xleftarrow{R} \mathbb{Z}_p$ and $h_b \leftarrow g^{s_b}$, $h_{1-b} \leftarrow c/h_b$
3. Choose $r_0, r_1 \xleftarrow{R} \mathbb{Z}_p$ and set $ct_b \leftarrow (g^{r_b}, m_b \oplus t_b)$

   $ct_{1-b} \leftarrow (g^{r_{1-b}}, t_{1-b})$ where $t_b, t_{1-b} \xleftarrow{R} \{0,1\}^n$ and $H(h_b^{r_b}) \mapsto t_b$

Claim: Under the CDH assumption and modeling H as a random oracle:

$$S(1^\lambda, b, m_b) \overset{c}{\approx} \text{View}_2((m_0, m_1), b)$$

To see this, observe that simulated view is identical unless distinguisher queries random oracle on input $h_{1-b}^{r_{1-b}}$. We use such a distinguisher to break CDH:

1. On input a CDH challenge (g, g^x, g^y).
2. Set $c = g^x$. Sample $s_b \xleftarrow{R} \mathbb{Z}_p$, $h_b \leftarrow g^{s_b}$ and $h_{1-b} \leftarrow c/g^{s_b}$.
3. Choose $r_b \xleftarrow{R} \mathbb{Z}_p$ and set $ct_b \leftarrow (g^{r_b}, m_b \oplus t_b)$ where $t_b \xleftarrow{R} \{0,1\}^n$ and $H(h_b^{r_b}) \mapsto t_b$.
4. Set $ct_{1-b} \leftarrow (g^y, t_{1-b})$ where $t_{1-b} \xleftarrow{R} \{0,1\}^n$

Perfect simulation of real/simulated views, unless distinguisher queries random oracle at $h_{1-b}^y = g^{xy}/g^{s_b y}$, in which case, we can compute $g^{xy} = h_{1-b}^y \cdot (g^y)^{s_b}$ and break CDH.

Receiver Security: Sender's view in the protocol consists of two uniformly random group elements h_0, h_1 such that $h_0 h_1 = c$. Simulator just needs to sample $h_0 \xleftarrow{R} \mathbb{G}$ and set $h_1 \leftarrow c/h_0$. This is a perfect simulation.

General idea: Sender sends a challenge. Receiver chooses a single ElGamal public/secret keypair for message it wants to decrypt. This uniquely defines the other public key (and receiver is not able to compute the secret key efficiently). Sender then encrypts both messages and receiver is able to decrypt exactly one of them. Other message hidden by semantic security of ElGamal.

Naor-Pinkas OT (without random oracles): Let $\mathbb{G}$ be a prime order group.

sender $(m_0, m_1 \in \mathbb{G})$ receiver $(b \in \{0,1\})$

$x \xleftarrow{R} \mathbb{Z}_p \quad y \xleftarrow{R} \mathbb{Z}_p \quad h \leftarrow g^x \quad u \leftarrow g^y$

$v_b \leftarrow g^{xy} \quad v_{1-b} \xleftarrow{R} \mathbb{G} \setminus \{g^{xy}\}$

$\xleftarrow{(g,\, h,\, u,\, v_0,\, v_1)}$

$\alpha_0, \beta_0 \xleftarrow{R} \mathbb{Z}_p$

$ct_0 \leftarrow (u^{\alpha_0} g^{\beta_0},\ v_0^{\alpha_0} h^{\beta_0} m_0)$

$\alpha_1, \beta_1 \xleftarrow{R} \mathbb{Z}_p$

$ct_1 \leftarrow (u^{\alpha_1} g^{\beta_1},\ v_1^{\alpha_1} h^{\beta_1} m_1)$

$\xrightarrow{ct_0,\ ct_1}$

$m_b \leftarrow \dfrac{ct_{b,1}}{ct_{b,0}^x}$

Correctness:

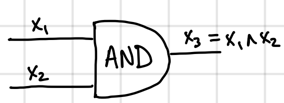
$$\left.\begin{aligned} ct_{b,1} &= v_b^{\alpha_b} h^{\beta_b} m_b = g^{\alpha_b xy} h^{\beta_b} m_b = g^{(\alpha_b y + \beta_b)x} m_b \\ ct_{b,0} &= u^{\alpha_b} g^{\beta_b} = g^{\alpha_b y + \beta_b} \end{aligned}\right\} \quad \frac{ct_{b,1}}{ct_{b,0}^x} = m_b$$

Sender Security: We will argue that m_{1-b} is perfectly hidden. Since $v_{1-b} \neq g^{xy}$, $(u_{1-b}^{\alpha_{1-b}} g^{\beta_{1-b}},\ v_{1-b}^{\alpha_{1-b}} h^{\beta_{1-b}})$ are uniformly random (see DDH random self reduction). Thus, m_{1-b} is perfectly hidden by $v_{1-b}^{\alpha_{1-b}} h^{\beta_{1-b}}$ (over the sender randomness $\alpha_{1-b}, \beta_{1-b}$). Simulator just chooses uniformly random pair for ct_{1-b}.

Receiver Security: Follows by DDH in $\mathbb{G}$. In particular, by DDH, v_b is computationally indistinguishable from uniformly random group element, so can construct simulator that just outputs random group elements (independent of b).

Yao's Protocol for Secure 2-Party Computation

Key ingredient: "garbling" protocol (garbled circuits)

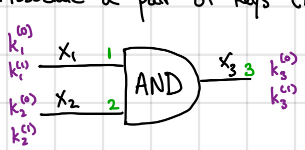


$\Longrightarrow$

truth table:

x_1	x_2	$x_1 \wedge x_2$
0	0	0
0	1	0
1	0	0
1	1	1

1) Associate a pair of keys $(k_i^{(0)}, k_i^{(1)})$ with each wire i in the circuit



$k_i^{(b)}$: key associated with wire value b for wire i [symmetric encryption key]

2) Prepare garbled truth table for the gate

↳ Replace each entry of truth table with corresponding key

↳ Encrypt output key with each of the input keys

x_1		x_2		$x_3 = x_1 \wedge x_2$	
0	$k_1^{(0)}$	0	$k_2^{(0)}$	0	$k_3^{(0)}$
0	$k_1^{(0)}$	1	$k_2^{(1)}$	0	$k_3^{(0)}$
1	$k_1^{(1)}$	0	$k_2^{(0)}$	0	$k_3^{(0)}$
1	$k_1^{(1)}$	1	$k_2^{(1)}$	1	$k_3^{(1)}$

$\Longrightarrow$

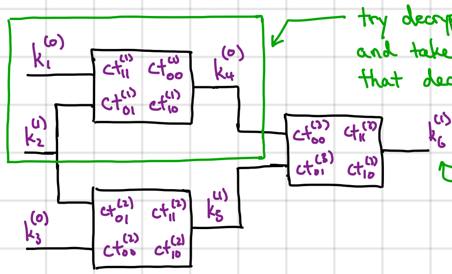
$$\left.\begin{aligned} ct_{00} &\leftarrow \mathrm{Encrypt}(k_1^{(0)}, \mathrm{Encrypt}(k_2^{(0)}, k_3^{(0)})) \\ ct_{01} &\leftarrow \mathrm{Encrypt}(k_1^{(0)}, \mathrm{Encrypt}(k_2^{(1)}, k_3^{(0)})) \\ ct_{10} &\leftarrow \mathrm{Encrypt}(k_1^{(1)}, \mathrm{Encrypt}(k_2^{(0)}, k_3^{(0)})) \\ ct_{11} &\leftarrow \mathrm{Encrypt}(k_1^{(1)}, \mathrm{Encrypt}(k_2^{(1)}, k_3^{(1)})) \end{aligned}\right\} \text{randomly shuffle ciphertexts}$$

3) Construct decoding table for output values

$k_3^{(0)} \mapsto 0$
$k_3^{(1)} \mapsto 1$ } Alternatively, can just encrypt output values instead of keys for output wires

<u>General garbling transformation</u>: construct garbled table for each gate in the circuit, prepare decoding table for each output wire in the circuit

<u>Evaluating a garbled circuit</u>:



try decrypting each ciphertext with the input keys, and take the output key to be the ciphertext that decrypts

decode using decoding table

<u>Invariant</u>: given keys for input wires of a gate, can derive key corresponding to output wire $\Longrightarrow$ enables gate-by-gate evaluation of garbled circuit
↳ <u>Requirement</u>: Evaluator needs to obtain keys (labels) for its inputs (but without revealing which set of labels it requested)

<u>Abstractly</u>: $\text{Garble}(1^\lambda, C) \to (\tilde{C}, \{L_{i,b}\}_{i \in [n], b \in \{0,1\}})$ ← number of input wires
$\text{Eval}(\tilde{C}, \{L_{i,x_i}\}_{i \in [n]}) \to y$

- <u>Correctness</u>: For all circuits $C : \{0,1\}^n \to \{0,1\}^m$ and all $x \in \{0,1\}^n$:
if $(\tilde{C}, \{L_{i,b}\}_{i \in [n], b \in \{0,1\}}) \leftarrow \text{Garble}(1^\lambda, C)$,
$$\Pr[\text{Eval}(\tilde{C}, \{L_{i,x_i}\}_{i \in [n]}) = C(x)] = 1$$
- <u>Security</u>: There exists an efficient simulator S such that for all circuits $C : \{0,1\}^n \to \{0,1\}^m$ and $x \in \{0,1\}^n$:
for $(\tilde{C}, \{L_{i,b}\}_{i \in [n], b \in \{0,1\}}) \leftarrow \text{Garble}(1^\lambda, C)$:
$$\{(\tilde{C}, \{L_{i,x_i}\}_{i \in [n]})\} \overset{c}{\approx} S(1^\lambda, C, C(x))$$
← can also consider notion where only $|C|$ is provided to S

Namely, the garbled circuit and <u>one set</u> of labels can be simulated just given the output $C(x)$.

We can show that Yao's garbling transformation satisfies above definition. [There are also other types of garbling schemes.]

Yao's garbled circuit protocol:

circuit C

Alice (garbler) ↳ private input x

Bob (evaluator) ↳ private input y

Alice: 1. Prepare garbled circuit for C

Bob: 1. Prepare OT queries for bits of y

Bob → Alice: OT for labels for y

Alice: 2. Prepare OT responses for Bob's inputs. Messages correspond to wire labels.

Alice → Bob: OT responses for labels of Bob's input

Alice → Bob: garbled circuit

Alice → Bob: labels for Alice's input x

Bob: 2. Evaluate garbled circuit to learn $C(x,y)$

Correctness: Follows by correctness of OT and of the garbling construction

Security: Relies on security of OT and garbling transformation

↳ Simulate Bob's view given output of computation (using the garbled circuit simulator) ← relies on OT simulator to simulate OT responses

↳ Simulate Alice's view using OT simulator

Variants:
1. If both parties should learn output, Bob can send it to Alice.
2. If Alice and Bob should learn distinct outputs, Alice can have the functionality output a blinded / encrypted version of her output.
3. Can extend to malicious security (need additional rounds and some modifications).