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 <u>true</u>. This week, we consider a generalization to two-party computation:

Alice (x) $\longrightarrow$ $\longleftarrow$ $\longrightarrow$ Bob (y)

$\downarrow$ $f_1(x,y)$ $\qquad$ $\downarrow$ $f_2(x,y)$

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

<u>Examples</u>: Yao's millionaire problem: Alice and Bob are millionaires and they want to learn which one of them is richer <u>without</u> 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.

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.

Party 1's output $\qquad$ Party 2's output

Let $f = (f_1, f_2)$ be a two-party functionality, and let π be an interactive protocol for computing f.

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

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

The protocol π should satisfy the following properties:

- <u>Correctness</u>: For all inputs x, y:

$$\Pr[\mathrm{output}_i^\pi(x,y) = f_i(x,y)] = 1.$$

- <u>(Semi-Honest) Security</u>: There exist efficient simulators S_1 and S_2 such that for all inputs x and y

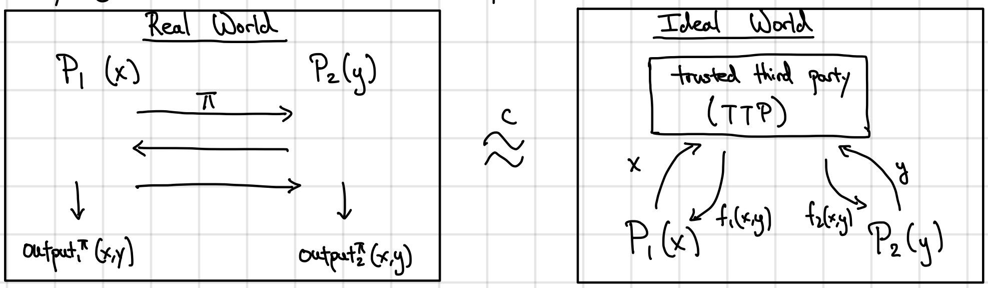
$$\{S_1(x, f_1(x,y)), f(x,y)\} \overset{c}{\approx} \{\mathrm{view}_1^\pi(x,y), \mathrm{output}^\pi(x,y)\}$$

$$\{S_2(y, f_2(x,y)), f(x,y)\} \overset{c}{\approx} \{\mathrm{view}_2^\pi(x,y), \mathrm{output}^\pi(x,y)\}$$

<u>Notes</u>:
- Security definition says that the view of each party can be <u>simulated</u> just given the party's input and its output in the computation (i.e., the <u>minimal</u> 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 <u>not</u> 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 <u>honest</u> 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

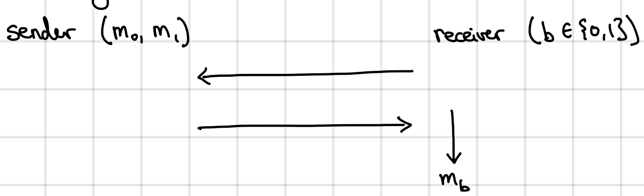
[Imagine a secure auction where adversary learns results first and decides to abort the protocol and claim "network failure" before honest parties can obtain the results]

- 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 $m_0, m_1 \in \{0,1\}^n$:

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

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

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

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 $b \in \{0,1\}$,

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

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

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 $(b \in \{0,1\})$

$c \xleftarrow{R} \mathbb{G}$ $\xrightarrow{\quad c \quad}$

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

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

check that $h_0 h_1 = c$

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

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

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

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

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

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

$$S(b, m_b) \overset{c}{\approx} \mathrm{view}_2((m_0, m_1), b)$$

we have not formally introduced the random oracle model so will give idea only:

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 $t_b = H(h_b^{r_b})$
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 adversary evaluates random oracle at $h_{1-b}^y = g^{xy}/g^{s_b y}$, in which case, the adversary can also compute $g^{xy} = h_{1-b}^y \cdot g^{s_b y}$ [formally, the random oracle allows us to extract the value of h_{1-b}^y, and solve 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 <u>perfect</u> 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.

Can also construct 2-message OT <u>without</u> random oracles from DDH (Naor-Pinkas)

↳ Many other constructions also possible – OT is a core building block in crypto, and in particular, <u>complete</u> for MPC