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:

\[
\begin{align*}
\text{Alice (x)} & \quad \text{Bob (y)} \\
\downarrow & \quad \downarrow \\
\downarrow & \quad \downarrow \\
\rightarrow & \quad \rightarrow \\
\text{Output (f(x,y))} & \quad \text{Output (f(x,y))}
\end{align*}
\]

[Alice has a (secret) input \( x \) and Bob has (secret) input \( y \). 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 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.
- 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 \( f = (f_1, f_2) \) be a two-party functionality, and let \( \pi \) be an interactive protocol for computing \( f \).

1. We write \( \text{view}_i(x,y) \) to denote the view of party \( i \in \{1,2\} \) on a protocol execution \( \pi \) on inputs \( x \) and \( y \). Note that \( \text{view}_i(x,y) \) is a random variable containing party \( i \)’s input, randomness, and all of the messages party \( i \) received during the protocol execution.
2. We write \( \text{output}_i(x,y) \) to denote the output of protocol \( \pi \) on inputs \( x \) and \( y \). We will write \( \text{output}_i(x,y) = (\text{output}_1(x,y), \text{output}_2(x,y)) \) to refer to the outputs of the respective parties. The value \( \text{output}_i(x,y) \) can be computed from \( \text{view}_i(x,y) \).

The protocol \( \pi \) should satisfy the following properties:

- **Correctness:** For all inputs \( x,y \):
  \[
  \Pr[\text{output}_{1}(x,y) = f_1(x,y)] = 1.
  \]

- **(Semi-Honest) Security:** There exist efficient simulators \( S_1 \) and \( S_2 \) such that for all inputs \( x \) and \( y \):
  \[
  \{S_1(x, f_1(x,y)), f(x,y)\} \simeq \{\text{view}_{1}(x,y), \text{output}_{2}(x,y)\} \\
  \{S_2(y, f_2(x,y)), f(x,y)\} \simeq \{\text{view}_{2}(x,y), \text{output}_{1}(x,y)\}
  \]

**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 any other party’s input is hidden. Only true if \( f \) does not leak the other party’s input.
- Definition only requires simulating the views 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.
Defining security against malicious adversaries is not easy. Here is a sketch (informal) of how it is typically done:

### Security

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).

### Fairness

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)

### Our Focus

Semi-honest two-party computation.

- This is necessary and sufficient for general multiparty computation (MPC).

### Key Cryptographic Building Blocks: Oblivious Transfer (OT)

**Sender** $(m_0, m_1)$

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

- Receiver 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.

### Correctness

For all messages $m_0, m_1 \in \{0,1\}$:

- $\Pr[\text{output}^x((m_0, m_1), b) = (1, m_0)] = 1$

### Sender Security

There exists an efficient simulator $S$ such that for all $m_0, m_1 \in \{0,1\}$, $b \in \{0,1\}$

- $S(b, m_b) \approx \text{view}_x((m_0, m_1), b)$

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

### Receiver Security

There exists an efficient simulator $S$ such that for all $m_0, m_1 \in \{0,1\}$ and $b \in \{0,1\}$

- $S(m_0, m_1) \approx \text{view}_x((m_0, m_1), b)$

Sender's view can be simulated just given its input messages $m_0, m_1$. (receiver's choice bit $b$ is hidden).
**Bellare-H. OT**: Let \( G \) be a prime order group and \( H: G \rightarrow \{0,1\}^n \) be a hash function (modeled as a random oracle):

- **Sender** \( (m, m_s \in \{0,1\}^n) \)
  
  \[
  \begin{aligned}
  c &\leftarrow G \\
  h &\leftarrow g^c \\
  s_b &\leftarrow Z_p \\
  h_b &\leftarrow g^{s_b} \\
  h_1 &\leftarrow g^{s_b} \\
  \end{aligned}
  \]

  check that \( h h_1 = c \)

- **Receiver** \( (b \in \{0,1\}) \)

  \[
  \begin{aligned}
  r &\leftarrow Z_p \\
  r_b &\leftarrow g^r \\
  c_b &\leftarrow (g^r, h_b \oplus m) \\
  \end{aligned}
  \]

**Correctness**: By construction, \( C_{tb} = (g^r)^x = h_b^{s_b} \) and correctness follows.

**Sender Security**: We construct simulator as follows. On input \((b, m_b)\):

1. Choose \( c \leftarrow G \)

2. Choose \( s_b \leftarrow Z_p \) and \( h_b \leftarrow g^{s_b} \), \( h_1 \leftarrow g^{s_b} \)

3. Choose \( r, r_1 \leftarrow Z_p \) and set \( c_b \leftarrow (g^{r_1}, m_b \oplus t_b) \) where \( t_b = H(h_b^x) \) and \( c_1 \leftarrow (g^{r_1}, t_1) \).

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

**General idea**: Sender sends a challenge. Receiver chooses a single ElGamal public/private 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 without random oracles from DDH (Naor-Pinkas)**

- Many other constructions also possible. OT is a core building block in crypto, and in particular, complete for MPC.