Modeling Adversaries

- **Adversarial behavior**
  - **Semi-honest**: follows the protocol specification
    - Tries to learn more than allowed by inspecting transcript
  - **Malicious**: follows any arbitrary strategy

- **Adversarial power**
  - **Polynomial-time**
  - **Computationally unbounded**: information-theoretic security

(based on slides of Yehuda Lindell)
Modeling Adversaries

- Corruption strategy
  - **Static:** the set of corrupted parties is fixed before the execution begins
  - **Adaptive:** the adversary can corrupt parties during the execution, based on what has happened
    - Models modern “hacking”
    - In general, **much harder**!
Execution Setting

- **Stand-alone**
  - Consider a single protocol execution only (or that only a single execution is under attack)

- **Concurrent general composition**
  - Arbitrary protocols executed concurrently
  - Realistic setting, very important model

- **Stand-alone vs composition**
  - **Stand-alone**: a good place to start studying secure computation, techniques and tools are helpful
  - **Composition**: true goal for constructions
Notations:

- Security parameter $n$
- We wish security to hold for all inputs of all lengths, as long as $n$ is large enough

Function $\mu$ is negligible: if for every polynomial $p(\cdot)$ there exists an $N$ such that for all $n > N$ we have $\mu(n) < 1/p(n)$
Preliminaries

- Probability ensemble $X = \{X(a,n)\}$
  - Infinite series, indexed by a string $a$ and natural $n$

- Each $X(a,n)$ is a random variable
  - In our context: the output of a protocol execution with input $a$ and security parameter $n$
  - Probability space: randomness of parties
Computational indistinguishability $X \approx Y$

For every (non-uniform) polynomial-time distinguisher $D$ there exists a negligible function $\mu$ such that for every $a$ and all large enough $n$'s:

$$|\Pr[D(X(a,n))=1] - \Pr[D(Y(a,n))=1]| < \mu(n)$$
Notation

- **Functionality**
  - \( f = (f_1, f_2) \): for input vector \( x \), each \( f_i(x) \) is a random variable (for probabilistic functionalities)
  - Party \( P_i \) receives \( f_i \)
  - We denote \( (x, y) \rightarrow (f_1(x,y), f_2(x,y)) \)
Semi-Honest Adversaries

- **Simulation:**
  - Given input and output, can generate the adversary’s view of a protocol execution
  - Important: since parties follow protocol, the inputs are **well defined**
Security definition: Semi-Honest Adversaries

∀ semi-honest adversary A controlling P1, ∃ simulator S1 such that for every pair of inputs \((x,y)\), the following are computationally indistinguishable
- The output of A, and the output of the honest party P2 after a protocol execution
- The output of S1 given \(x_1\) and \(f_1(x,y)\), and the value \(f_2(x,y)\)

Similarly, ∀ semi-honest A controlling P2, ∃ S2, such that ∀ inputs \((x,y)\), the following are computationally indistinguishable
- The output of A, and the output of the honest party P1 after a protocol execution
- The output of S2 given \(x_2\) and \(f_2(x,y)\), and the value \(f_1(x,y)\)
Semi-Honest Adversaries

\[ f_1(x, y) \text{ and transcript} \quad f_2(x, y) \]
Semi-Honest Adversaries

$\text{x}$

Protocol

$\text{Simulator}$

$f_1(x, y)$ & transcript

April 8, 2014
Properties

- Correctness, independence of inputs, fairness are all non-issues in the semi-honest model

- Why is privacy guaranteed by this definition?
  - If the adversary can compute something after a real protocol execution, it can compute it just from the input/output
  - The adversary’s view in an execution can be generated from the input and output only
  - Very similar to zero-knowledge
Joint Distribution

- A crucial point: need to consider the *joint distribution* of adversary’s output and honest parties’ output

- In the definition:
  - We compare the distribution of all inputs and outputs together with the adversary’s output
Joint Distribution

- **Example:**
  - **Functionality:** A outputs random bit, B outputs nothing
    - B should clearly not learn A’s output bit
  - **Protocol:** A chooses a random bit, outputs it, and sends the bit to B (who ignores it)

- This protocol is clearly insecure.
  - But it is simulatable when separately looking at the distribution of B’s view and actual outputs
  - However, it is not simulatable when working according to the definition
Deterministic Functionalities

- In the case of deterministic functionalities, the outputs are fully determined by the inputs
- It suffices to *separately* prove
  - Correctness
  - Simulation: show that can generate view of semi-honest adversary (corrupted parties’ view), given inputs and outputs only

- In other words...
Separately prove the following two statements

- The output of the protocol is indistinguishable from the output of the functionality.

  There exists a simulator $S_1$ such that for any adversary $A$ controlling $P_1$, the output of $A$, and the output of $S_1$ given $x_1$ and $f_1(x)$, are indistinguishable.

  Similarly, that there exists a simulator $S_2$ such that for any adversary $A$ controlling $P_2$, the output of $A$, and the output of $S_2$ given $x_2$ and $f_2(x)$, are indistinguishable.
Malicious Adversaries

- **First attempt:** require the existence of a simulator that generates the adversary’s view given the inputs/outputs of the corrupted party

- **Problem:** what are the inputs used by the adversary?
  - They are not necessarily those written on the input tape
  - They are not explicit: the adversary doesn’t run the protocol but arbitrary code
  - For example, in the Bellare-Micali OT protocol, a malicious server can send two random messages without knowing what they encrypt
What is the best we could hope for?
- An incorruptible trusted party
- All parties send inputs to trusted party (over perfectly secure communication lines)
- Trusted party computes output
- Trusted party sends each party its output (over perfectly secure communication lines)
- This is an ideal world

What can an adversary do?
- Just choose its input…
The Ideal/Real Paradigm

- We would like our real protocol to behave like the ideal world
- Formalizing this notion:
  - For every adversary $A$ attacking the real protocol, there exists an adversary $S$ in the ideal model such that the output distributions (of all parties) are computationally indistinguishable
  - $S$ simulates a real protocol execution while interacting in the ideal world
  - Here we always look at the joint output distribution
The Ideal/Real Paradigm

Real World

Protocol

Ideal World

Trusted Party

f(x', y)

arbitrary output

output

arbitrary output

f(x', y)

April 8, 2014
“Formal” Security Definition

- Protocol \( \pi \) securely computes a function \( f \) if:
  - For every non-uniform polynomial-time real-model adversary \( A \), there exists a non-uniform polynomial-time ideal-model adversary \( S \), such that for all input vectors and auxiliary inputs:
    - the joint outputs of \( A \) and the honest party in a real execution of \( \pi \) are indistinguishable from the joint outputs of \( S \) and the honest party in an ideal execution where the trusted party computes \( f \).
Properties

- The following properties hold

  - Privacy: from adversary’s outputs
  - Correctness: from honest party’s output
  - Independence of inputs: from ideal execution
  - Fairness and guaranteed output delivery: from ideal execution
Relaxing the Ideal Model

- In some cases, this ideal model is too strong and cannot be achieved

- **Fairness** cannot be achieved in general without an honest majority
  - Consider two parties and consider removing the last message of the protocol execution
    - Works for coin tossing…
In order to model the case that fairness is not guaranteed, change the instructions of the trusted party in the ideal model:

- Trusted party receives input from all parties
- Trusted party sends corrupted party’s output to adversary
- Adversary says “continue” or “halt”
- If “continue”, trusted party sends output to honest party; else, it sends “abort”