Many thanks to Vitaly Shmatikov of the University of Texas, Austin for providing these slides.
Motivation

- General framework for describing computation between parties who do not trust each other

- Example: elections
  - N parties, each one has a “Yes” or “No” vote
  - Goal: determine whether the majority voted “Yes”, but no voter should learn how other people voted

- Example: auctions
  - Each bidder makes an offer
    - Offer should be committing! (can’t change it later)
  - Goal: determine whose offer won without revealing losing offers
More Examples

▶ Example: distributed data mining
  - Two companies want to compare their datasets without revealing them
    - For example, compute the intersection of two lists of names

▶ Example: database privacy
  - Evaluate a query on the database without revealing the query to the database owner
  - Evaluate a statistical query on the database without revealing the values of individual entries
  - Many variations
A Couple of Observations

◆ In all cases, we are dealing with distributed multi-party protocols
  • A protocol describes how parties are supposed to exchange messages on the network

◆ All of these tasks can be easily computed by a trusted third party
  • The goal of secure multi-party computation is to achieve the same result without involving a trusted third party
How to Define Security?

- Must be mathematically rigorous
- Must capture all realistic attacks that a malicious participant may try to stage
- Should be “abstract”
  - Based on the desired “functionality” of the protocol, not a specific protocol
  - Goal: define security for an entire class of protocols
Functionality

◆ K mutually distrustful parties want to jointly carry out some task

◆ Model this task as a function

\[ f: (\{0,1\}^*)^K \rightarrow (\{0,1\}^*)^K \]

◆ Assume that this functionality is computable in probabilistic polynomial time
Ideal Model

- Intuitively, we want the protocol to behave “as if” a trusted third party collected the parties’ inputs and computed the desired functionality.
  - Computation in the ideal model is secure by definition!
A protocol is secure if it *emulates* an ideal setting where the parties hand their inputs to a "trusted party," who locally computes the desired outputs and hands them back to the parties.

\[ f_1(x_1, x_2) \]

\[ f_2(x_1, x_2) \]

[Goldreich-Micali-Wigderson 1987]
Adversary Models

- Some of protocol participants may be corrupt
  - If all were honest, would not need secure multi-party computation
- Semi-honest (aka passive; honest-but-curious)
  - Follows protocol, but tries to learn more from received messages than she would learn in the ideal model
- Malicious
  - Deviates from the protocol in arbitrary ways, lies about her inputs, may quit at any point
- For now, we will focus on semi-honest adversaries and two-party protocols
Correctness and Security

◆ How do we argue that the real protocol “emulates” the ideal protocol?

◆ Correctness
  • All honest participants should receive the correct result of evaluating function f
    – Because a trusted third party would compute f correctly

◆ Security
  • All corrupt participants should learn no more from the protocol than what they would learn in ideal model
  • What does corrupt participant learn in ideal model?
    – His input (obviously) and the result of evaluating f
Simulation

- Corrupt participant’s view of the protocol = record of messages sent and received
  - In the ideal world, view consists simply of his input and the result of evaluating $f$

- How to argue that real protocol does not leak more useful information than ideal-world view?

- Key idea: simulation
  - If real-world view (i.e., messages received in the real protocol) can be simulated with access only to the ideal-world view, then real-world protocol is secure
  - Simulation must be indistinguishable from real view
Technicalities

◆ **Distance** between probability distributions A and B over a common set X is

\[ \frac{1}{2} \times \sum_X (|\Pr(A=x) - \Pr(B=x)|) \]

◆ **Probability ensemble** $A_i$ is a set of discrete probability distributions
  
  ● Index $i$ ranges over some set $I$

◆ **Function** $f(n)$ is **negligible** if it is asymptotically smaller than the inverse of any polynomial

\[ \forall \text{ constant } c \ \exists m \text{ such that } |f(n)| < \frac{1}{n^c} \ \forall n > m \]
Notions of Indistinguishability

- Simplest: ensembles $A_i$ and $B_i$ are equal
- Distribution ensembles $A_i$ and $B_i$ are statistically close if $\text{dist}(A_i, B_i)$ is a negligible function of $i$
- Distribution ensembles $A_i$ and $B_i$ are computationally indistinguishable ($A_i \approx B_i$) if, for any probabilistic polynomial-time algorithm $D$, $|\Pr(D(A_i) = 1) - \Pr(D(B_i) = 1)|$ is a negligible function of $i$
  - No efficient algorithm can tell the difference between $A_i$ and $B_i$ except with a negligible probability
SMC Definition (1st Attempt)

- Protocol for computing \( f(X_A, X_B) \) betw. A and B is secure if there exist efficient simulator algorithms \( S_A \) and \( S_B \) such that for all input pairs \( (x_A, x_B) \) ...

- Correctness: \( (y_A, y_B) \approx f(x_A, x_B) \)
  - Intuition: outputs received by honest parties are indistinguishable from the correct result of evaluating \( f \)

- Security: \( \text{view}_A(\text{real protocol}) \approx S_A(x_A, y_A) \)
  \( \text{view}_B(\text{real protocol}) \approx S_B(x_B, y_B) \)
  - Intuition: a corrupt party’s view of the protocol can be simulated from its input and output

- This definition does not work! Why?
Randomized Ideal Functionality

- Consider a coin flipping functionality
  \[ f() = (b, -) \] where \( b \) is random bit
  - \( f() \) flips a coin and tells A the result; B learns nothing

- The following protocol “implements” \( f() \)
  1. A chooses bit \( b \) randomly
  2. A sends \( b \) to B
  3. A outputs \( b \)

- It is obviously insecure \( \text{(why?)} \)
- Yet it is correct and simulatable according to our attempted definition \( \text{(why?)} \)
SMC Definition

- Protocol for computing $f(X_A, X_B)$ between A and B is secure if there exist efficient simulator algorithms $S_A$ and $S_B$ such that for all input pairs $(x_A, x_B)$ ...

- Correctness: $(y_A, y_B) \approx f(x_A, x_B)$

- Security: $(\text{view}_A(\text{real protocol}), y_B) \approx (S_A(x_A, y_A), y_B)$  
  $(\text{view}_B(\text{real protocol}), y_A) \approx (S_B(x_B, y_B), y_A)$
  
  Intuition: if a corrupt party’s view of the protocol is correlated with the honest party’s output, the simulator must be able to capture this correlation

- Does this fix the problem with coin-flipping $f$?
Oblivious Transfer (OT) [Rabin 1981]

- Fundamental SMC primitive

- A inputs two bits, B inputs the index of one of A’s bits
- B learns his chosen bit, A learns nothing
  - A does not learn which bit B has chosen; B does not learn the value of the bit that he did not choose
- Generalizes to bitstrings, M instead of 2, etc.
One-Way Trapdoor Functions

- Intuition: one-way functions are easy to compute, but hard to invert (skip formal definition for now)
  - We will be interested in one-way permutations

- Intuition: one-way trapdoor functions are one-way functions that are easy to invert given some extra information called the trapdoor
  - Example: if $n=pq$ where $p$ and $q$ are large primes and $e$ is relatively prime to $\varphi(n)$, $f_{e,n}(m) = m^e \mod n$ is easy to compute, but it is believed to be hard to invert
  - Given the trapdoor $d = e^{-1} \mod \varphi(n)$, $f_{e,n}(m)$ is easy to invert because $f_{e,n}(m)^d = (m^e)^d = m \mod n$
Hard-Core Predicates

Let $f: S \rightarrow S$ be a one-way function on some set $S$

B: $S \rightarrow \{0,1\}$ is a **hard-core predicate** for $f$ if

- $B(x)$ is easy to compute given $x \in S$
- If an algorithm, given only $f(x)$, computes $B(x)$ correctly with prob $> \frac{1}{2} + \varepsilon$, it can be used to invert $f(x)$ easily
  - Consequence: $B(x)$ is hard to compute given only $f(x)$
- Intuition: there is a bit of information about $x$ s.t. learning this bit from $f(x)$ is as hard as inverting $f$

Goldreich-Levin theorem

$B(x,r) = r \cdot x$ is a hard-core predicate for $g(x,r) = (f(x),r)$

- $f(x)$ is any one-way function, $r \cdot x = (r_1x_1) \oplus \ldots \oplus (r_nx_n)$
Oblivious Transfer Protocol

Assume the existence of some family of one-way trapdoor permutations

A chooses his input $i$ (0 or 1)

B chooses a one-way permutation $F$ and corresponding trapdoor $T$

Chooses random $r_0, r_1, x, y_{not\ i}$

Computes $y_i = F(x)$

Computes $m_i \oplus (r_i \cdot x)$

$$= (b_i \oplus (r_i \cdot T(y_i))) \oplus (r_i \cdot x)$$

$$= (b_i \oplus (r_i \cdot T(F(x)))) \oplus (r_i \cdot x) = b_i$$
Proof of Security for B

A \hspace{10cm} B

- Chooses random $r_{0,1}$, $x$, $y_{\text{not } i}$
- Computes $y_i = F(x)$
- \hspace{5cm} $r_0, r_1, y_0, y_1$
- Computes $m_i \oplus (r_i \cdot x)$

$y_0$ and $y_1$ are uniformly random regardless of A's choice of permutation $F$ (why?). Therefore, A's view is independent of B's input $i$. 

$\text{Proof of Security for B}$
Proof of Security for A (Sketch)

- Need to build a simulator whose output is indistinguishable from B’s view of the protocol.

**Sim**

- Chooses random $F$, random $r_0, r_1$, $x$, $y_{not \ i}$
- Computes $y_i = F(x)$
- Sets $m_i = b_i \oplus (r_i \cdot T(y_i))$, random $m_{not \ i}$

Knows $i$ and $b_i$ (why?)

**B**

- Random $r_{0,1}$, $x$, $y_{not \ i}$
- $y_i = F(x)$
- $r_0, r_1, y_0, y_1$

The only difference between simulation and real protocol:

In simulation, $m_{not \ i}$ is random (why?)

In real protocol, $m_{not \ i} = b_{not \ i} \oplus (r_{not \ i} \cdot T(y_{not \ i}))$
Proof of Security for A (Cont’d)

- Why is it computationally infeasible to distinguish random \( m \) and \( m' = b \oplus (r \cdot T(y)) \)?
  - \( b \) is some bit, \( r \) and \( y \) are random, \( T \) is the trapdoor of a one-way trapdoor permutation

- \( (r \cdot x) \) is a hard-core bit for \( g(x,r) = (F(x),r) \)
  - This means that \( (r \cdot x) \) is hard to compute given \( F(x) \)

- If \( B \) can distinguish \( m \) and \( m' = b \oplus (r \cdot x') \) given only \( y = F(x') \), we obtain a contradiction with the fact that \( (r \cdot x') \) is a hard-core bit
  - Proof omitted
Yao’s Protocol
Yao’s Protocol

- Compute any function securely
  - ... in the semi-honest model
- First, convert the function into a boolean circuit

![Diagram of boolean circuits and truth tables]

Truth table for AND:

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Truth table for OR:

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
1: Pick Random Keys For Each Wire

- Next, evaluate **one gate** securely
  - Later, generalize to the entire circuit

- Alice picks two **random keys** for each wire
  - One key corresponds to “0”, the other to “1”
  - 6 keys in total for a gate with 2 input wires

![Diagram of an AND gate with random keys labeled for Alice and Bob.]
2: Encrypt Truth Table

- Alice encrypts each row of the truth table by encrypting the output-wire key with the corresponding pair of input-wire keys.

Original truth table:

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Encrypted truth table:

- \(E_{k_{0x}}(E_{k_{0y}}(k_{0z}))\)
- \(E_{k_{0x}}(E_{k_{1y}}(k_{0z}))\)
- \(E_{k_{1x}}(E_{k_{0y}}(k_{0z}))\)
- \(E_{k_{1x}}(E_{k_{1y}}(k_{1z}))\)
3: Send Garbled Truth Table

Alice randomly permutes ("garbles") encrypted truth table and sends it to Bob. Does not know which row of garbled table corresponds to which row of original table.

Garbled truth table:

- $E_{k_0x}(E_{k_0y}(k_{0z}))$
- $E_{k_0x}(E_{k_1y}(k_{0z}))$
- $E_{k_1x}(E_{k_0y}(k_{0z}))$
- $E_{k_1x}(E_{k_1y}(k_{1z}))$
- $E_{k_0x}(E_{k_0y}(k_{0z}))$
- $E_{k_0x}(E_{k_1y}(k_{0z}))$

Diagram:
- Alice
- Bob
- X, Y, Z
- $k_{0x}, k_{1x}$
- $k_{0y}, k_{1y}$
- $k_{0z}, k_{1z}$
- AND gate
4: Send Keys For Alice’s Inputs

Alice sends the key corresponding to her input bit
- Keys are random, so Bob does not learn what this bit is

Alice sends $k_{0z}$, $k_{1z}$

Learns $K_{b'y}$ where $b'$ is Alice's input bit, but not $b'$ (why?)

Garbled truth table:

- $E_{k_1x}(E_{k_0y}(k_{0z}))$
- $E_{k_0x}(E_{k_1y}(k_{0z}))$
- $E_{k_1x}(E_{k_1y}(k_{1z}))$
- $E_{k_0x}(E_{k_0y}(k_{0z}))$

If Alice’s bit is 1, she simply sends $k_{1x}$ to Bob; if 0, she sends $k_{0x}$
Alice and Bob run oblivious transfer protocol

- Alice’s input is the two keys corresponding to Bob’s wire
- Bob’s input into OT is simply his 1-bit input on that wire

Garbled truth table:

- $E_{k_0x}(E_{k_0y}(k_{0z}))$
- $E_{k_0x}(E_{k_1y}(k_{0z}))$
- $E_{k_1x}(E_{k_0y}(k_{1z}))$
- $E_{k_1x}(E_{k_1y}(k_{1z}))$
- $E_{k_0x}(E_{k_0y}(k_{0z}))$

Run oblivious transfer
- Alice’s input: $k_{0y}, k_{1y}$
- Bob’s input: his bit $b$

Bob learns $k_{by}$

What does Alice learn?

Knows $K_{b'x}$ where $b'$ is Alice’s input bit and $K_{by}$ where $b$ is his own input bit
6: Evaluate Garbled Gate

- Using the two keys that he learned, Bob decrypts exactly one of the output-wire keys
  - Bob does not learn if this key corresponds to 0 or 1
    - Why is this important?

```
\[ \text{Garbled truth table:} \]

\[ E_{k_{0x}}(E_{k_{0y}}(k_{0z})) \]
\[ E_{k_{1x}}(E_{k_{0y}}(k_{0z})) \]
\[ E_{k_{1x}}(E_{k_{1y}}(k_{1z})) \]
\[ E_{k_{0x}}(E_{k_{0y}}(k_{0z})) \]
```

Suppose \( b' = 0, b = 1 \)

This is the only row Bob can decrypt. He learns \( K_{0z} \)

Knows \( K_{b'x} \) where \( b' \) is Alice’s input bit and \( K_{by} \) where \( b \) is his own input bit.
An Important Aside

Why is it that Bob can only decrypt one row of the garbled circuit?

- Use encryption scheme that has an *elusive range* and
- Use encryption scheme that has an *efficiently verifiable range*

Elusive Range: Roughly, the probability that an encryption under one key is in the range of an encryption under another key is negligible.

Efficiently Verifiable Range: A user, given a key, can efficiently verify whether ciphertext is in the range of that key.
Example (Lindell, Pinkas paper)

- $F = \{f_k\}$ a family of pseudorandom functions with $f_k: \{0,1\}^n \rightarrow \{0,1\}^{2n}$ for $k$ in $\{0,1\}^n$
- For $x$ in $\{0,1\}^n$, $r$ a random $n$ bit string, define $E_k(x) = (r, f_k(r) \text{ XOR } x_0^n)$
  - $x_0^n$ is the concatenation of $x$ and $n$ bit string of 0s
- Elusive range: the low order $n$ bits of $f_k(r)$ are revealed (and fixed) by the XOR.
  - The odds of having two keys giving that same low order $n$ bits is $1/2^n$
- Verifiable range: Given $r$ and a key $k$, it is trivial to verify that ciphertext is in the range of $E_k$. 
7: Evaluate Entire Circuit

◆ In this way, Bob evaluates entire garbled circuit
  • For each wire in the circuit, Bob learns only one key
  • It corresponds to 0 or 1 (Bob does not know which)
    – Therefore, Bob does not learn intermediate values (why?)

◆ Bob tells Alice the key for the final output wire and she tells him if it corresponds to 0 or 1
  • Bob does **not** tell her intermediate wire keys (why?)
Brief Discussion of Yao’s Protocol

- Function must be converted into a circuit
  - For many functions, circuit will be huge
- If $m$ gates in the circuit and $n$ inputs, then need $4m$ encryptions and $n$ oblivious transfers
  - Oblivious transfers for all inputs can be done in parallel
- Yao’s construction gives a constant-round protocol for secure computation of any function in the semi-honest model
  - Number of rounds does not depend on the number of inputs or the size of the circuit!
    - Though the size of the data transferred does!