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!
Slightly More Formally

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.

[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 he would learn in the ideal model
- Malicious
  - Deviates from the protocol in arbitrary ways, lies about his 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 (First Attempt)

- 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:** $(y_A, y_B) \approx f(x_A, x_B)$
  - 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 a 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 (why?)

- Yet it is correct and simulatable according to our attempted definition (why?)
SMC Definition

― 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: $(y_A, y_B) \approx f(x_A, x_B)$

― 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)

- **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.

[Rabin 1981]
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 \( \phi(n) \), \( f_{e,n}(m) = m^e \mod n \) is easy to compute, but it is believed to be hard to invert
  • Given the trapdoor \( d \) s.t. \( de=1 \mod \phi(n) \), \( f_{e,n}(m) \) 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:

1. \( B(x) \) is easy to compute given \( x \in S \).
2. 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 \( f(x) \).
3. 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, \( r \cdot x = (r_1 x_1) \oplus \ldots \oplus (r_n x_n) \).
Oblivious Transfer Protocol

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

A chooses his input i (0 or 1)

Chooses a one-way permutation F and corresponding trapdoor T

Computes yi = F(x)

B chooses random r0,1, x, y_{not i}

Computes y_i = F(x)

\[ b_0 \oplus (r_0 \cdot T(y_0)), b_1 \oplus (r_1 \cdot T(y_1)) \]

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

B

\[ F \]

Chooses random \( r_{0,1}, x, y_{\text{not } i} \)
Computes \( y_i = F(x) \)

\( r_0, r_1, y_0, y_1 \)

\[ b_0 \oplus (r_0 \cdot T(y_0)), b_1 \oplus (r_1 \cdot T(y_1)) \]

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

\( y_0 \text{ and } y_1 \) are uniformly random regardless of A's choice of permutation F (why?).
Therefore, A's view is independent of B's input i.
Proof of Security for A (Sketch)

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

- Chooses random $F$, random $r_{0,1}$, $x$, $y_{\text{not } i}$
- Computes $y_i = F(x)$
- Sets $m_i = b_i \oplus (r_i \cdot T(y_i))$
- Random $m_{\text{not } i}$

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

The only difference between simulation and real protocol:

- In simulation, $m_{\text{not } i}$ is random (why?)
- In real protocol, $m_{\text{not } i} = b_{\text{not } i} \oplus (r_{\text{not } i} \cdot T(y_{\text{not } i}))$
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