Session 1: Definitions and Oblivious Transfer

Yehuda Lindell
Bar-Ilan University
Secure Multiparty Computation

• A set of parties with private inputs
• Parties wish to jointly compute a function of their inputs so that certain security properties are preserved
• Properties must be ensured even if some of the parties maliciously attack the protocol
• Can model any cryptographic task
Applications

• Elections
• Auctions
• Private database search
• Privacy-preserving data mining
• Secure set intersection
• Much much more...
Security Requirements

• Consider a secure auction (with secret bids):
  – An adversary may wish to learn the bids of all parties
    – to prevent this, require PRIVACY
  – An adversary may wish to win with a lower bid than
    the highest – to prevent this, require CORRECTNESS
  – But, the adversary may also wish to ensure that it
    always gives the highest bid – to prevent this, require
    INDEPENDENCE OF INPUTS
  – An adversary may try to abort the execution if its bid
    is not the highest – require FAIRNESS
General Security Properties

• **Privacy:** only the output is revealed
• **Correctness:** the function is computed correctly
• **Independence of inputs:** parties cannot choose inputs based on others’ inputs
• **Fairness:** if one party receives output, all receive output
• **Guaranteed output delivery**
Defining Security

• Option 1: analyze security concerns for each specific problem
  – Auctions: as in previous slide
  – Elections: privacy, correctness and fairness only (?)

• Problems:
  – How do we know that all concerns are covered?
  – Definitions are application dependent and need to be redefined from scratch for each task
Defining Security

• Option 2: general definition that captures all (most) secure computation tasks

• Properties of any such definition
  – Well-defined adversary model
  – Well-defined execution setting
  – Security guarantees are clear and simple to understand
Modeling Adversaries

• Adversarial behavior
  – Semi-honest: follows the protocol specification
    • Tries to learn more than allowed by inspecting transcript
  – Malicious: follows any arbitrary strategy
  – Covert: follows any arbitrary strategy, but is averse to being caught...

• Adversarial power
  – Polynomial-time: computational security
  – Computationally unbounded: information-theoretic security
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”
    • Cannot use strategies that choose a small set of representatives to compute for all
    • 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
Feasibility of Secure Computation

• Assuming an honest majority, any functionality can be securely computed
  – Even information theoretically, and with adaptive security

• Without an honest majority, it is impossible to achieve fairness in general

• Without an honest majority, any functionality can be securely computed without fairness
Preliminaries

• **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 $p(n)$ there exists an $N$ such that for all $n > N$ we have $\mu(n) < \frac{1}{p(n)}$
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: output of protocol execution with input $a$ and security parameter $n$
    • Probability space: randomness of parties
Preliminaries

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

• Functionality
  – $f = (f_1, \ldots, f_m)$: 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
Semi-Honest Adversaries

• For every semi-honest $A$, there exists a simulator $S$ such that for every set of corrupted parties $I$ and every vector of inputs $x$, the following are computationally indistinguishable:

  – The output of $A$, and the outputs of all parties after a protocol execution.

  – The output of $S$ given $x_i$ and $f_i(x)$ for all $i \in I$, and all the values $f_1(x), \ldots, f_m(x)$.
Semi-Honest Adversaries

The REAL execution

Protocol

Arbitrary Output (w.l.o.g.: adversary’s view in execution)

\( f(x, y) \)

Simulation

Simulator Output

\( x, f(x, y) \)

\( f(x, y) \)
Properties

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

• Why is privacy guaranteed by this definition?
  – The adversary’s view in an execution can be generated from the input and output only
  – If the adversary can compute something after a real protocol execution, it can compute it just from the input/output
  – 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 is simulatable when separately looking at distribution of $B$’s view and actual outputs
Deterministic Functionalities

• In the case of deterministic functionalities, the outputs are fully determined by the inputs

• It suffices to separately prove
  – Correctness
  – Simulation: can generate view of semi-honest adversary (corrupted parties’ view), given inputs and outputs only
    • This is significantly easier!
Malicious Adversaries

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

• 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
Malicious Adversaries

• We also need to require independence of inputs, correctness, fairness etc.
  – These properties are not captured by “view simulation” alone

• Can we separate correctness and privacy?
  – Instead of computing $f$, compute a function that reveals first input bit of other party
  – Correctness or privacy???

• What about independence of inputs and privacy?
The Ideal/Real Paradigm

• 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

• The real protocol must be 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) 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

Ideal World

$x'$

Protocol

$f(x', y)$

Trusted Party

$y$

arbitrary output

output

arbitrary output

$f(x, y)$

Secure Computation and Efficiency
Bar-Ilan University, Israel 2015
“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 parties in a real execution of $\pi$ is computationally indistinguishable from the joint outputs of $S$ and the honest parties in an ideal execution where the trusted party computes $f$
Properties

• The following properties hold
  – **Privacy**: from adversary’s outputs
  – **Correctness**: from honest parties’ outputs
  – **Independence of inputs**: from ideal execution
  – **Fairness and guaranteed output delivery**: from ideal execution

  – More?
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
Relaxing the Ideal Model

• Change the instructions of the trusted party
  – Trusted party receives input from all parties
  – Trusted party sends corrupted parties’ outputs to adversary
  – Adversary says “continue” or “halt”
  – If “continue”, trusted party sends output to honest parties; else, it sends “abort”
Reactive Functionalities

• Functionalities that obtain inputs and provide outputs in stages

• Examples:
  – Mental poker
  – Commitment schemes

• This is also useful for relaxing ideal functionalities (give side information to $S$)

• The definition extends naturally to this as well
Advantages of This Approach

• General – it captures ALL applications
• The specifics of an application are defined by its functionality, security is defined as above
• The security guarantees achieved are easily understood (because the ideal model is easily understood)
  – We can be confident that we did not “miss” any security requirements
Using Secure Computation

• The ideal-model paradigm
  – You don’t need to understand anything about how a protocol works to use it
  – You just need to imagine an incorruptible trusted party computing the functionality for you

• Very advantageous for usage
Sequential Modular Composition

• Sequential modular composition:
  – Secure protocols are run sequentially, with arbitrary messages sent in between them

• Why consider this?
  – An important security goal within itself
  – Very helpful (if not crucial) tool for analyzing the security of protocols

• Formalization – Hybrid Model
  – A trusted party helps to compute a sub-functionality
  – REAL messages & IDEAL messages
Sequential Modular Composition

• Subprotocols $\rho_i$ securely compute functionalities $f_i$
• Protocol $\pi$ securely computes $g$ in a hybrid model where a trusted party is used to compute every $f_i$
  – This is much easier to analyze since each $f_i$ is effectively “perfectly secure”
• **Theorem:** assuming the above, the real protocol $\pi^\rho$ that uses real calls to each $\rho_i$ instead of a trusted party for $f_i$, securely computes $g$. 
Concurrent Composition

• We have considered the stand-alone model
  – This implies sequential composition

• What about concurrent composition?
  – An Internet-like setting where many (arbitrary, secure and insecure) protocols are run concurrently, with the adversary controlling the scheduling

• This models the real-world setting more accurately
  – We don’t know what the result is of running stand-alone protocols concurrently with related inputs
Concurrent Composition

• Concurrent general composition
  – Strictly harder than the stand-alone model
  – *Impossible* without some trusted set-up assumption (like a common reference string)

• The UC definition (universal composability) guarantees security in this setting
  – Efficient UC security is a special challenge...

• Recommended to study UC next, after studying the stand-alone setting
Relaxed Definitions

• In order to achieve high efficiency, sometimes can consider weaker definitions
  – Semi-honest (but this is very weak)
  – Covert adversaries: adversary may be malicious but is guaranteed to be caught cheating with good probability
    • Suitable where adversaries can be penalized for being caught cheating (e.g., business loss)
  – Privacy only (malicious)
    • Problematic...
Summary

• Semi-honest: simulator given input/output generates the adversary’s view
  – Probabilistic functionalities – must consider joint distribution of view and outputs
  – Deterministic functionalities: easier, suffices to separately consider correctness and view simulation

• Malicious: ideal-real simulation

• Sequential composition

• Advanced topics
  – Concurrent composition
  – Relaxed definition
  – Semi-honest vs malicious
General vs Specific Protocols

• Most of the school will focus on general protocols
  – Convert the function into a Boolean or arithmetic circuit
  – Compute the circuit securely

• It seems that for specific problems, specific protocols should be more secure
General vs Specific Protocols

• **General protocols – advantages**
  – Implement once
  – Very flexible: almost no difference between
    • Set intersection
    • Size of set intersection
    • Output 1 if set intersection size is greater than $k$
  – In many cases is competitive, and in fact the fastest solution known
OBLIVIOUS TRANSFER
Oblivious Transfer (OT)

Called 1-out-of-2 oblivious transfer ($OT_{1}^{2}$)
Fundamental Primitive

• **OT is complete**
  – If can compute OT then can compute any functionality

• **Constructing OT**
  – OT cannot be constructed from PKE in a black box manner
  – Can be constructed from
    • Enhanced trapdoor permutations
    • DDH, RSA, Lattices
Just a Few Important OT Results

• OT is symmetric

• Can construct efficient $OT^N_1$ and $OT^N_k$ from $OT^2_1$

• Can construct malicious OT from semi-honest OT in a black-box manner (inefficiently)

• Many variants of OT are equivalent
  – Random OT
  – Rabin OT
  – Weak OT
Efficient OT from DDH

• Recall the DDH assumption over a group $\mathbb{G}$ of order $q$ with generator $g$
  
  – The DDH assumption says that
    $\{(g, g^a, g^b, g^{ab})\} \approx \{(g, g^a, g^b, g^c)\}$
  
  where $a, b, c \leftarrow \mathbb{Z}_q$ are random
Semi-Honest OT

• Recall ElGamal encryption
  – Secret key: random $a \leftarrow \mathbb{Z}_q$
  – Public key: $h = g^a$
  – Encrypt $m \in \mathbb{G}$: $c = (u, v) = (g^r, h^r \cdot m)$, random $r \in \mathbb{Z}_q$
  – Decrypt $(u, v)$: compute $m = \frac{v}{u^a}$

• Note: $\frac{v}{u^a} = \frac{h^r \cdot m}{(g^r)^a} = \frac{h^r \cdot m}{(g^a)^r} = \frac{h^r \cdot m}{h^r} = m$
Semi-Honest OT

Choose $a_\sigma$; compute $h_\sigma = g^{a_\sigma}$
Choose random $h_{1-\sigma} \in \mathbb{G}$

Encrypt $x_0$ with $h_0$
Encrypt $x_1$ with $h_1$

$u_0, v_0 = (g^r, (h_0)^r \cdot x_0)$
$u_1, v_1 = (g^s, (h_1)^s \cdot x_1)$

Note:
- Encrypt $x_0$ with $h_0$
- Encrypt $x_1$ with $h_1$
Semi-Honest OT – Security

- **Security:**
  - Alice sees only two public keys, which are two random group elements (and so learns nothing about $\sigma$)
    - Formally, simulate by sending two random group elements
  - Bob knows only one private key and so learns only $x_\sigma$
    - Formally, simulate by encrypting $x_\sigma$ with $h_\sigma$, and encrypting garbage (e.g., 0) with $h_{1-\sigma}$

```
Choose $a_\sigma$; compute $h_\sigma = g^{a_\sigma}$
Choose random $h_{1-\sigma} \in \mathbb{G}$

Encrypt $x_0$ with $h_0$  
Encrypt $x_1$ with $h_1$

$h_0, h_1$  
$c_0, c_1$

Decrypt $c_\sigma$ with $a_\sigma$
```
More Efficient Semi-Honest OT

Choose \( a_\sigma \); compute \( h_\sigma = g^{a_\sigma} \)
Choose random \( h_{1-b} \in \mathbb{G} \)

Choose \( r \leftarrow \mathbb{Z}_q \)
Compute \( u = g^r \)

Compute \( v_0 = (h_0)^r \cdot x_0 \)
Compute \( v_1 = (h_1)^r \cdot x_1 \)

\( u, v_0, v_1 \)

Output \( x_\sigma = \frac{v_\sigma}{u^{a_\sigma}} \)
Malicious Adversaries

• **Corrupted sender:**
  – Sender cannot cheat
  – Simulator can “extract” both $x_0, x_1$ by choosing both $h_0$ and $h_1$ so that it knows the secret keys

• **Corrupted receiver:**
  – Receiver can choose both $h_0$ and $h_1$ so that it knows the secret keys
Preventing Malicious

• The idea:
  – Alice sends a random group element $w$
  – Bob chooses $h_0, h_1$ so that $h_0 \cdot h_1 = w$
    • Bob can easily do this by choosing $a_\sigma$, computing $h_\sigma = g^{a_\sigma}$ and setting $h_{1-\sigma} = w/h_\sigma$
    • Bob cannot know both DLOGs of $h_0, h_1$ or it can compute the DLOG of $H$

• Encryption uses a random oracle since “not completely knowing” a secret key doesn’t suffice
  – Encrypt by $(g^r, HASH((h_0)^r) \oplus x_0),...$
State of the Art – OT

• Semi-honest adversaries
  – Receiver: 2 exponentiations + send 2 group elements
  – Sender: 3 exponentiations + send 3 group elements

• Malicious adversaries (Random Oracle)
  – Same as semi-honest

• Malicious adversaries (PVW)
  – Receiver: 3 exponentiations + send 2 group elements
  – Sender: 8 exponentiations (effectively 6) + send 4 group elements
Proving Malicious Security

• Proving security in the malicious model is tricky and subtle

• The ideal/real model paradigm
  – Need a simulator who internally runs the real adversary and externally interacts with the trusted party (sending input and getting output)
  – The simulator needs to “extract” the real adversary’s input, get output, and make the output match

• We demonstrate the ideal/real proof technique for the problem of coin tossing
Proving Malicious Security

• **Blum’s protocol** (with ElGamal):
  - Party $P_1$:
    - Choose random $b \in \{0,1\}$ and $r, s \leftarrow \mathbb{Z}_q$
    - Compute $h = g^r$, $u = g^s$, $v = h^s \cdot g^b$
    - Send $(h, u, v)$ to $P_2$
  - Party $P_2$:
    - Choose random $b' \in \{0,1\}$
    - Send $b'$ to $P_1$
  - Party $P_1$ sends $r, s, b$ to $P_2$
  - Party $P_2$ verifies that $h = g^r$, $u = g^s$, $v = h^s \cdot g^b$
  - Both parties output $b \oplus b'$
Intuition

• Consider a corrupt $P_2$
  – By the security of El Gamal encryption, it knows nothing about $b$ when it chooses $b'$

• Consider a corrupt $P_1$
  – The values $(h, u, v)$ fully define $b$
    • There exists a single pair $(r, s)$ so that $h = g^r, u = g^s$
    • The value $v$ can either be $h^s$ or $h^s \cdot g$, but not both
  – $P_2$ chooses $b'$ after $P_1$ sends $b$; by the above, $P_1$ cannot change $b$ and so $P_1$ cannot bias the output
Proving Security - $P_1$ corrupted

- Let $A$ be an adversary; $S$ works as follows
- $S$ receives a random bit $\beta$ from the trusted party
- $S$ invokes $A$ and receives $(h, u, v)$
- $S$ works as follows:
  - $S$ internally hands $A$ the value $b' = 0$
  - $S$ rewinds $A$ and internally hands $A$ the value $b' = 1$
  - If $A$ replies correctly both times, $S$ learns the value $b$, sets $b' = b \oplus \beta$, and outputs this as $A$’s view. In addition, $A$ externally sends continue to the TTP
  - If $A$ does not reply correctly either time, $S$ sends abort to the TTP and outputs a random $b'$ as $A$’s view
  - If $A$ aborts once, then $S$ learns the value $b$, sets $b' = b \oplus \beta$, and outputs this as $A$’s view. If $A$ aborts on this $b'$ then $S$ sends abort to the TTP; else it sends continue to the TTP
Proving Security - $P_2$ corrupted

- Let $A$ be an adversary; $S$ works as follows
- $S$ receives a random bit $\beta$ from the trusted party
- $S$ invokes $A$ and works as follows:
  - $S$ chooses a random $b$ and internally hands $A$ the tuple $(h, u, v)$ computed correctly for $b$
  - $S$ receives $b'$ from $A$
  - If $b \oplus b' = \beta$ then $S$ outputs $(h, u, v)$ and $(r, s, b)$ as its view, and sends continue to the TTP
  - Else, $S$ rewinds $A$ and goes to the beginning again
- Note: there is no abort here since we can just take $b' = 0$ as default if $P_2$ doesn’t respond
Summary

• Oblivious transfer is a fundamental primitive
  – It is heavily used in most general secure computation protocols

• Oblivious transfer is very efficient
  – But it does cost exponentiations every time!
  – This afternoon we will see how to improve this