ECE 443/518 – Computer Cyber Security
Lecture 21 Secure Multi-Party Computation

Professor Jia Wang
Department of Electrical and Computer Engineering
Illinois Institute of Technology

October 29, 2025

Outline

Reading Assignment

- This lecture: Secure Multi-Party Computation
- Next lecture: Garbled Circuit

Outline

Oblivious Transfer (OT)

- Alice runs a pay-per-view service that provides access to n messages $m_1, m_2, \ldots, m_n$.
- Bob would like to access a particular message m_k.
- Bob don't want to let Alice know what is k.
  - For privacy reasons.
- Bob don't want to pay Alice a lot of money to obtain all the messages in order to hide k.
- Let's consider the simple case for two messages ($n = 2$).
  - Alice's secret: m_1, m_2.
  - Bob's secret: $k \in \{1, 2\}$.
  - At the end, Bob learns m_k but not the other among the two messages, and Alice learns nothing about k.
- How could this even be possible?
  - Assume Alice and Bob are <u>honest but curious</u>.

Mechanism Design

- Alice's RSA key pair: $k_{pr} = (n = pq, d)$, $k_{pub} = (n, e)$.

1. Alice sends Bob two random messages x_1 and x_2.
2. Bob generates a random message y and sends Alice v.
   - $v = (y^e + x_k) \bmod n$.
3. Alice sends Bob m'_1 and m'_2.
   - $m'_1 = m_1 + ((v - x_1)^d \bmod n)$.
   - $m'_2 = m_2 + ((v - x_2)^d \bmod n)$.
4. Bob computes $m'_k - y$ to recover m_k.
   - For $k = 1$, RSA guarantees that $m'_1 = m_1 + ((v - x_1)^d \bmod n) = m_1 + (y^{ed} \bmod n) = m_1 + y$.
   - Same applies when $k = 2$.
   - So Bob indeed learns m_k.

Analysis for Alice

- The only piece of information Alice directly learns from Bob is the message v.
  - $v = (y^e + x_k) \bmod n$.
  - Note that Alice has no kwowledge about y and k.
- With x_1 and x_2, Alice may derive y_1 and y_2.
  - $y_1 = (v - x_1)^d \bmod n$.
  - $y_2 = (v - x_2)^d \bmod n$.
- $v \equiv y_1^e + x_1 \equiv y_2^e + x_2 \pmod{n}$.
  - Alice cannot decide which of y_1 and y_2 is y.
- Alice learns nothing about Bob's secret k.
  - No matter how powerful Alice is.

Analysis for Bob

- Assume $k = 1$ for Bob.
  - Bob will learn m_1.
  - Does Bob learn anything about m_2?
- Bob learns x_1, x_2, m'_1, m'_2 directly from Alice.
  - x_1 and x_2 are simply random messages, providing no information on m_2.
  - $m'_1 = m_1 + y$, having nothing to do with m_2.
- $m'_2 \equiv m_2 + (v - x_2)^d \equiv m_2 + (y^e + x_1 - x_2)^d \pmod{n}$.
  - If Bob is able to learn m_2, then Bob learns $m'_2 - m_2$ which is $(y^e + x_1 - x_2)^d$.
  - Since Bob also knows $y^e + x_1 - x_2$, this implies Bob is able to decrypt $y^e + x_1 - x_2$ into $(y^e + x_1 - x_2)^d$ without knowing d.
  - Since Alice chooses x_1 and x_2 arbitrarily, to decrypt $y^e + x_1 - x_2$ means Bob could decrypt any message encrypted without knowing d – this breaks RSA.
- Bob, if computationally bounded, learns nothing about m_2.

Outline

Oblivious Transfer (OT)

Secure Multi-Party Computation

Idea of Garbled Circuit

Secure Multi-Party Computation

- Assume there are n honest-but-curious parties $1, 2, \ldots, n$.
- Each party k possesses a secret value v_k.
- Together they compute $f = F(v_1, v_2, \ldots, v_n)$.
  - For a well-known function F.
- Confidentiality: secret remains secret.
  - Any party k should only learn f from the computation, but nothing more about secrets of other parties.
- Ignore integrity issues.

Examples: Voting

- Secret from every party: 0 or 1
- F computes the summation.
- Every party learns only f, the number of 1's.
- A party may learn exactly what other parties vote, e.g.
  - When there is only two parties, both know.
  - When $f = 0$ or n, everyone knows.
  - When $f = 1$ or $n - 1$, who votes 1 or 0 knows.

Examples: Salary Comparison

- Secret from every party: a number representing salary.
- F computes the maximum.
- Every party learns only f, the highest salary.
- If there are only two parties Alice and Bob,
  - Alice, if earns more, won't learn Bob's salary.
  - What if Alice run the salary comparison multiple times, each with a different number? Then she may know Bob's salary!
- Mechanism for secure multi-party computation should prevent evaluating F multiple times without consent from all parties.
  - A party is not able to change its secret when evaluating F.

Outline

Oblivious Transfer (OT)

Secure Multi-Party Computation

Idea of Garbled Circuit

Secure Two-Party Computation

- Let's consider two parties for simplicity.
- How could you represent arbitrary computations?

Circuit

- Encode secrets from Alice and Bob, as well as the result f from the computation, all as binary strings.
- F then becomes a boolean function.
  - Implemented as a boolean circuit.
- In particular, a combinational circuit.
  - Whose size is proportional to the effort to compute F.
  - We will not distinguish F from its combinational circuit implementation.

Example: NAND

- Secret from Alice: $a \in \{0, 1\}$
- Secret from Bob: $b \in \{0, 1\}$
- Can they compute $f = NAND(a, b)$ without revealing their own secrets?
  - If we could further extend this to any input bits and any number of NAND gates, then we could handle arbitrary combinational circuits.
- Note that for $f = NAND(a, b)$, if Bob chooses $b = 1$ then he can learn a from f.
  - This is allowed per definition of secure multi-party computation.
  - Not a concern if Bob chooses $b = 0$, or the circuit is much more complicated.

Idea of Garbled Circuit

- A collaboration between Alice and Bob.
- The garbler Alice garbles the circuit.
  - By encrypting every wire and every gate.
  - Send Bob the garbled circuit.
  - Send Bob her input bits (encrypted).
- Alice also helps Bob to encrypt his input bits.
  - So Bob is not able to change them and evaluate the circuit multiple times in order to learn Alice's input bits.
  - But what prevents Alice to learn Bob's input bits? How could Alice encrypts bits without knowing it?
- Then the evaluator Bob evalutes the garbled circuit.
  - Compute with encrypted boolean values.
- Finally Bob communicate with Alice to reveal the output bits.

Encrypting Wires

- For any wire W, Alice generates a random selection bit S_w.
- Then, Alice generates two random binary strings W_0 and W_1.
  - W_0 represents signal 0 and starts with S_w.
  - W_1 represents signal 1 and starts with $1 - S_w$.
  - Essentially S_w prevents Bob to learn which each random string represents but still provide some useful structure.
- Alice can tell what signal a binary string represents by inspecting its first bit.
- For the circuit $O = NAND(A, B)$, there are three wires.

Wire	Selection Bit	0	1
O	S_O	$O_0 = S_O \cdots$	$O_1 = (1 - S_O) \cdots$
A	S_A	$A_0 = S_A \cdots$	$A_1 = (1 - S_A) \cdots$
B	S_B	$B_0 = S_B \cdots$	$B_1 = (1 - S_B) \cdots$

Encrypting Wires (Cont.)

- For example, let's use 5 bits for each wire.

Wire	Selection Bit	0	1
O	$S_O = 1$	$O_0 = 10001 = 17$	$O_1 = 00101 = 5$
A	$S_A = 0$	$A_0 = 00110 = 6$	$A_1 = 10000 = 16$
B	$S_B = 1$	$B_0 = 10010 = 18$	$B_1 = 00010 = 2$

- Alice cannot send Bob the above table.
  - Otherwise Bob can evaluate the circuit multiple times with different signals to learn Alice's input bits.
- Bob need to calculate O_f from A_a and B_b.
  - Recall that $f = NAND(a, b)$

Discussions

Wire	Selection Bit	0	1
O	$S_O = 1$	$O_0 = 10001 = 17$	$O_1 = 00101 = 5$
A	$S_A = 0$	$A_0 = 00110 = 6$	$A_1 = 10000 = 16$
B	$S_B = 1$	$B_0 = 10010 = 18$	$B_1 = 00010 = 2$

- When completed, Bob sees about half of the table.
  - Bob learns one binary string per wire, e.g A_a, B_b, and O_f.
  - But Bob should not be able to learn the selection bits except for his input and the final output.
- Alice should prevent Bob to guess other binary strings and selection bits in the table correctly.
  - With m bits, Bob has a chance of $\frac{1}{2^m}$ to guess both the binary string and the selection bit correctly for each wire.
  - A very small chance even for a single wire if m is large enough.
  - Work on Homework 3 to see cases when Bob cannot guess them no matter what, using an argument similar to OTP.
- How could Bob calculate O_f from A_a and B_b?
  - For *NAND* in our example or more generally other gates.

Summary

- Oblivious transfer (OT) as a building block for more complicated protocols.
- Secure two-party computation via garbled circuit.