Outline

1. Secure Function Evaluation
   - Introduction
   - Examples
   - Malicious Setting

2. Oblivious Transfer
   - Definition
   - Examples

3. Garbled Circuits
   - Introduction
   - Garbled Circuits
   - Correctness

Secure Function Evaluation

Multi-Party Computation

$n$ players $P_i$ want to jointly evaluate $y_i = f_i(x_1, \ldots, x_n)$, for public functions $f_i$ so that

- $x_i$ is the private input of $P_i$
- $P_i$ eventually learns $y_i = f_i(x_1, \ldots, x_n)$
- ... and nothing else about $x_j$ for $j \neq i$

Security Notions

- Privacy
- Correctness
- Fairness (much harder to get)
Secure Function Evaluation

\textbf{t-Privacy}

If \( t \) parties collude, they cannot learn more on the other inputs than from their own/known inputs and outputs.

Note that the knowledge of \( y_i \) can leak some information on the \( x_j \)'s.

\textbf{Security Models}

- **Honest-but-curious**: all the players follow the protocol honestly, but the adversary knows all the inputs/outputs from \( t \) users.
- **Malicious users**: the adversary controls a fixed set of \( t \) players.
- **Dynamic adversary**: the adversary dynamically chooses the (up to) \( t \) players it controls.

Electronic Voting

\textbf{Private Evaluation of the Sum}

For all \( i: x_i \in \{0, 1\} \) and \( f_i(x_1, \ldots, x_n) = \sum_j x_j \)

\textbf{Example (Homomorphic Encryption)}

- \( P_i \) encrypts \( C_i = E(x_i) \) with an additively homomorphic encryption scheme.
- They all compute \( C = E(\prod x_i) \) and
- They jointly decrypt \( C \) to get \( y = \sum x_i \) using a distributed decryption.

Electronic Voting

\textbf{Privacy: Limitations}

In case of unanimity (i.e. \( \sum x_i = n \)), one learns all the \( x_i \)'s, even in the honest-but-curious setting.

This is not a weakness of the protocol, but of the functionality: one should just reveal the winner.

\textbf{Replay Attacks}

A malicious adversary could try to amplify \( P_1 \)'s vote, replaying its message \( C_1 \) by \( t \) corrupted players: this can leak \( P_1 \)'s vote \( x_1 \).

This can be avoided with non-malleable encryption.
Yao Millionaires’ Problem

Alice owns an integer \( x \) and Bob owns an integer \( y \),

in the end, they both learn whether \( x \leq y \) or not

Theorem

Given \( x = x_{n-1} \ldots x_0, y = y_{n-1} \ldots y_0 \in \{0, 1\}^n \), and denoting

\[
T_x^1 = \{x_{n-1} \ldots x_i | x_i = 1\} \quad T_y^0 = \{y_{n-1} \ldots y_i+1 | y_i = 0\}
\]

\[
x > y \iff T_x^1 \cap T_y^0 \neq \emptyset
\]

\[
x > y \iff \exists i < n, (x_i > y_i) \land (\forall j > i, x_j = y_j)
\]

\[
\iff \exists i < n, (x_i = 1) \land (y_i = 0) \land (\forall j < i, x_j = y_j)
\]

\[
\iff \exists i < n, (y_i = 0) \land (n-1 \ldots x_i = y_{n-1} \ldots y_i+1)
\]

\[
\iff |T_x^1 \cap T_y^0| = 1
\]
1 Secure Function Evaluation
   - Introduction
   - Examples
   - Malicious Setting

2 Oblivious Transfer

3 Garbled Circuits

GMW Compiler

- Commitment of the inputs
- Secure coin tossing
- Zero-knowledge proofs of correct behavior

GMW Compiler

[Goldreich-Micali-Wigderson – STOC 1987]

Secure 2-Party Computation

The 2-party particular case: on Alice's input $x$ and Bob's input $y$, Alice gets $f(x, y)$ and Bob gets $g(x, y)$, but nothing else.

Oblivious Transfer

[Alice owns two values $x_0, x_1$ and Bob owns a bit $b \in \{0, 1\}$, so that in the end, Bob learns $x_b$ and Alice gets nothing: $x = (x_0, x_1)$ and $y = b$, then $f((x_0, x_1), b) = x_b$ and $g((x_0, x_1), b) = \perp$]

Oblivious Transfer is equivalent to Secure 2-Party Computation

From an Oblivious Transfer Protocol, one can implement any 2-Party Secure Function Evaluation.

[Oblivious Transfer is equivalent to Secure 2-Party Computation - Kilian – STOC 1988]
Outline

1 Secure Function Evaluation

2 Oblivious Transfer
   - Definition
   - Examples

3 Garbled Circuits

Oblivious Transfer

Example (Bellare-Micali’s Construction – 1992)
In a discrete logarithm setting \((G, g, p)\), for \(x_0, x_1 \in G\)
- Alice chooses \(c \overset{R}{\leftarrow} G\) and sends it to Bob
- Bob chooses \(k \overset{R}{\leftarrow} \mathbb{Z}_p\), sets \(pk_b \leftarrow g^k\) and \(pk_{1-b} \leftarrow c/pk_b\), and sends \((pk_0, pk_1)\) to Alice
- Alice checks \(pk_0 \cdot pk_1 = c\) and encrypts \(x_i\) under \(pk_i\) (for \(i = 0, 1\)) with ElGamal:
  \[C_i \leftarrow g^{r_i} \text{ and } C'_i \leftarrow x_i \cdot pk_i^{r_i}, \text{ for } r_i \overset{R}{\leftarrow} \mathbb{Z}_p\]
- Bob can decrypt \((C_b, C'_b)\) using \(k\)

Because of the random \(c\) (unknown discrete logarithm),
Bob should not be able to infer any information about \(x_{1-b}\)
This is provably secure in the honest-but-curious setting

Example (Naor-Pinkas Construction – 2000)
In a discrete logarithm setting \((G, g, p)\), for \(x_0, x_1 \in G\)
- Bob chooses \(r, s, t \overset{R}{\leftarrow} \mathbb{Z}_p\), sets \(X \leftarrow g^r, Y \leftarrow g^s, Z_b \leftarrow g^{rs}, Z_{1-b} \leftarrow g^t\), and sends \((X, Y, Z_0, Z_1)\) to Bob
- Alice checks \(Z_0 \neq Z_1\) and re-randomizes the tuples:
  \[T_0 \leftarrow (X, Y_0' = Y^{u_0} g^{v_0}, Z_0' = Z_0^{u_0} X^{v_0}) \text{ and } T_1 \leftarrow (X, Y_1' = Y^{u_1} g^{v_1}, Z_1' = Z_1^{u_1} X^{v_1}), \text{ for } u_0, v_0, u_1, v_1 \overset{R}{\leftarrow} \mathbb{Z}_p\]
- Alice encrypts \(x_i\) under \(T_i\): \(C_i = Y_i'\) and \(C'_i = x_i \cdot Z_i'\)
- Bob can decrypt \((C_b, C'_b)\) using \(r\)

The re-randomization keeps the DH-tuple \(T_b\),
but perfectly removes information in \(T_{1-b}\)
This is provably secure in the malicious setting
Boolean circuit, Alice's inputs \((x_1, x_2, x_3)\), and Bob's inputs \((y_1, y_2, y_3)\):

They both learn \(z\) in the end, but nothing else.

### Garbled Gates

Alice converts the circuit into a generic circuit: 1-input or 2-input gates.

**1-Input Garbled Gate**

For the gate \(A\) (not):

\[
A = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}
\]

\[A = 0 \quad : C_A^0 = \text{Encrypt}(I_A^0, O_A^1)\]

\[A = 1 \quad : C_A^1 = \text{Encrypt}(I_A^1, O_A^0)\]

**2-Input Garbled Gate**

For the gate \(B\) (and):

\[
B = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}
\]

\[B = 00 \quad : C_B^{00} = \text{Encrypt}(I_B^0||J_B^0, O_B^0)\]

\[B = 01 \quad : C_B^{01} = \text{Encrypt}(I_B^1||J_B^0, O_B^0)\]

\[B = 10 \quad : C_B^{10} = \text{Encrypt}(I_B^0||J_B^1, O_B^0)\]

\[B = 11 \quad : C_B^{11} = \text{Encrypt}(I_B^1||J_B^1, O_B^0)\]
**Alice’s Inputs**

Alice publishes the ciphertexts in random order for each gate

Alice publishes the keys corresponding to her inputs:
- for \( x_1 \), she sends \( I^{x_1}_A \)
- for \( x_2 \), she sends \( J^{x_2}_B \)
- for \( x_3 \), she sends \( J^{x_3}_C \)

**Bob’s Inputs**

\[
\begin{align*}
Y_1 & \rightarrow A & Y_A \\
A = 1 & : C^0_A = \text{Encrypt}(I^0_A, O^1_A) & C^1_A = \text{Encrypt}(I^1_A, O^0_A)
\end{align*}
\]

**Oblivious Transfer**

Alice owns \( I^0_A, I^1_A \) and Bob owns \( y_1 \in \{0, 1\} \)
- Using an OT, Bob gets \( I^{y_1}_A \), while Alice learns nothing
- From the ciphertexts \( (C^b_b)_b \), Bob gets \( O^{y_A}_A \)

**Bob’s Inputs**

\[
\begin{align*}
A & \leftarrow B & Y_B \\
B = 0 & : C^{00}_B = \text{Encrypt}(I^0_B||J^0_B, O^0_B) & C^{01}_B = \text{Encrypt}(I^0_B||J^1_B, O^0_B) \\
C^{10}_B = \text{Encrypt}(I^1_B||J^0_B, O^0_B) & C^{11}_B = \text{Encrypt}(I^1_B||J^1_B, O^0_B)
\end{align*}
\]

**Internal Garbled Gates**

**Oblivious Transfer**

Alice owns \( I^0_B, I^1_B \) and Bob owns \( y_2 \in \{0, 1\} \)
- Using an OT, Bob gets \( I^{y_2}_B \), while Alice learns nothing
- Bob additionally knows \( J^{y_2}_B \)
- From the ciphertexts \( (C^{bb'}_B)_{bb'} \), Bob gets \( O^{y_A}_B \)

**Internal Garbled Gate**

For the gate \( E \) (or): 2 new random secret keys \( O^0_E, O^1_E \)
while \( I^0_E \leftarrow O^1_A, I^1_E \leftarrow O^0_A, J^0_E \leftarrow O^1_B, J^1_E \leftarrow O^0_B \)

\[
\begin{align*}
E = 0 & : C^{00}_E = \text{Encrypt}(I^0_E||J^0_E, O^0_E) & C^{01}_E = \text{Encrypt}(I^0_E||J^1_E, O^1_E) \\
C^{10}_E = \text{Encrypt}(I^1_E||J^0_E, O^1_E) & C^{11}_E = \text{Encrypt}(I^1_E||J^1_E, O^1_E)
\end{align*}
\]
Evaluation of Internal Gates

\[
E = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}:
\begin{align*}
C_{E}^{00} &= \text{Encrypt}(I_{A}^{0}||J_{B}^{0}, O_{E}^{0}) \\
C_{E}^{01} &= \text{Encrypt}(I_{A}^{0}||J_{B}^{1}, O_{E}^{1}) \\
C_{E}^{10} &= \text{Encrypt}(I_{A}^{1}||J_{B}^{0}, O_{E}^{1}) \\
C_{E}^{11} &= \text{Encrypt}(I_{A}^{1}||J_{B}^{1}, O_{E}^{1})
\end{align*}
\]

Evaluation of Gate E

Bob knows \( I_{E}^{y_A} = O_{E}^{y_A} \) and \( J_{B}^{y_E} = O_{B}^{y_E} \)

From the ciphertexts \((C_{E}^{bb'})_{bb'}\), Bob gets \( O_{E}^{y_E} \)

Output Garbled Gates

\[
G = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}:
\begin{align*}
C_{G}^{00} &= \text{Encrypt}(I_{G}^{0}||J_{G}^{0}, 0) \\
C_{G}^{01} &= \text{Encrypt}(I_{G}^{0}||J_{G}^{1}, 1) \\
C_{G}^{10} &= \text{Encrypt}(I_{G}^{1}||J_{G}^{0}, 1) \\
C_{G}^{11} &= \text{Encrypt}(I_{G}^{1}||J_{G}^{1}, 1)
\end{align*}
\]

Evaluation of Gate G

Bob knows \( I_{E}^{y_E} = O_{E}^{y_E} \) and \( J_{G}^{y_F} = O_{F}^{y_F} \)

From the ciphertexts \((C_{G}^{bb'})_{bb'}\), Bob gets \( z \in \{0, 1\} \)

Bob can then transmit \( z \) to Alice

Outline

1. Secure Function Evaluation
2. Oblivious Transfer
3. Garbled Circuits
   - Introduction
   - Garbled Circuits
   - Correctness
The previous construction assumes that

- Bob extracts the correct plaintext among the multiple candidates
  \[ \Rightarrow \text{Redundancy is added to the plaintext} \]
  (or authenticated encryption)

They have to trust each other

- Alice correctly builds garbled gates: the ciphertexts are correct
  \[ \Rightarrow \text{Cut-and-choose technique} \]

- Alice plays the oblivious transfer protocols with correct inputs
  \[ \Rightarrow \text{Inputs are committed, checked during the cut-and-choose,} \]
  and ZK proofs are done during the OT

- Bob sends back the correct value $z$
  \[ \Rightarrow \text{Random tags are appended to the final results 0 and 1} \]
  that Bob cannot guess