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

*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.

**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

**Private Evaluation of the Sum**

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

**Example (Homomorphic Encryption)**

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

Electronic Voting

**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.

**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.
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

### Equality Test

Alice owns a value $x \in [A, B]$ and Bob owns a value $y \in [A, B]$, in the end, they both learn whether $x = y$ or not

#### With Homomorphic Encryption

- Alice encrypts $C = E(x)$ with an additively homomorphic encryption scheme
- Bob computes $C^0 = E(r(x - y))$, for a random element $r$
- Alice computes $C^{0\oplus} = E(r^0(x - y))$, for a random element $r^0$
- They jointly decrypt $C^{0\oplus}$: the value is 0 iff $x = y$ (or random)

### Yao Millionaires’ Problem

Alice owns a value $x \in [A, B]$ and Bob owns a value $y \in [A, B]$, in the end, they both learn whether $x = y$ or not

#### With Homomorphic Encryption

- Alice encrypts $C_i = E(X_i)$ with an additively homomorphic encryption scheme
- Bob computes $C^0_i = E(r_i(X_i - Y_i))$, for random elements $r_i$ and sends them in random order
- Alice computes $C^{0\oplus}_i = E(r^0_i(X_i - Y_i))$, for random elements $r^0_i$
- They jointly decrypt the $C^{0\oplus}_i$’s: one value is 0 iff $x > y$

---

**Yao Millionaires’ Problem**

Alice owns an integer $x \in [0, 2^n]$ and Bob owns an integer $y \in [0, 2^n]$, in the end, they both learn whether $x \leq y$ or not

**Theorem** [Lin-Tzeng – 2005]

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

\[
T^1_x = \{x_{n-1} \ldots x_i | x_i = 1\} \quad T^0_y = \{y_{n-1} \ldots y_{i+1} | y_i = 0\}
\]

\[
x \leq y \iff \exists i, T^1_x \cap T^0_y \neq \emptyset
\]

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

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

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

\[
x \leq y \iff |T^1_x \cap T^0_y| = 1
\]
### Outline

1. Secure Function Evaluation
   - Introduction
   - Examples
   - Malicious Setting
2. Oblivious Transfer
3. Garbled Circuits

### GMW Compiler

<table>
<thead>
<tr>
<th>[Goldreich-Micali-Wigderson – STOC 1987]</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Commitment of the inputs</td>
</tr>
<tr>
<td>- Secure coin tossing</td>
</tr>
<tr>
<td>- Zero-knowledge proofs of correct behavior</td>
</tr>
</tbody>
</table>

### 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 $g((x_0, x_1), b) = \perp$](Rabin - 1981)

**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**

[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 $g((x_0, x_1), b) = \perp$](Killian - STOC 1988)
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 \(c \leftarrow G\) and sends it to Bob
- Bob chooses \(k \leftarrow \mathbb{Z}_p\), 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:
  \[C_i \leftarrow g^{r_i} \text{ and } C_i^0 \leftarrow x_i \cdot pk_i^{r_i}, \text{ for } r_i \leftarrow \mathbb{Z}_p\]
- Bob can decrypt \((C_b, C_b^0)\) 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 \(r, s, t \leftarrow \mathbb{Z}_p\), 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:
  \[T_0 \leftarrow (X, Y_0 = Y^{u_0} g^{v_0}, Z_0 = Z_0^{u_0} X^{v_0})\]  and
  \[T_1 \leftarrow (X, Y_1 = Y^{u_1} g^{v_1}, Z_1 = Z_1^{u_1} X^{v_1})\], for \(u_0, v_0, u_1, v_1 \leftarrow \mathbb{Z}_p\)
- Alice encrypts \(x_i\) under \(T_i\): \(C_i = Y_i^0\) and \(C_i^0 = x_i \cdot Z_i^0\)
- Bob can decrypt \((C_b, C_b^0)\) 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

Boolean circuit, Alice's inputs \((x_1, x_2, x_3)\), and Bob's inputs \((y_1, y_2, y_3)\):

\[
\begin{align*}
\text{A} &= \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 1 \end{bmatrix} \quad \text{not} \quad C = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix} \quad \text{or} \\
\text{B} &= \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 1 \end{bmatrix} \quad \text{and} \\
\text{D} &= \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix} \quad \text{line} \\
\text{E} &= \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 1 \end{bmatrix} \quad \text{or} \\
\text{F} &= \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 1 \end{bmatrix} \quad \text{and} \\
\text{G} &= \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 1 \end{bmatrix} \quad \text{or}
\end{align*}
\]

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

Outline

1. Secure Function Evaluation
2. Oblivious Transfer
3. Garbled Circuits
   - Introduction
   - Garbled Circuits
   - Correctness

Garbled Gates

Alice generates the garbled gates

1-Input Garbled Gate
For the gate A (not): 4 random secret keys \(I^0_A, I^1_A, O^0_A, O^1_A\)

\[
A = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 1 \end{bmatrix} : C^0_A = \text{Encrypt}(I^0_A, O^1_A) \quad C^1_A = \text{Encrypt}(I^1_A, O^0_A)
\]

2-Input Garbled Gate
For the gate B (and): 8 random secret keys \(I^0_B, I^1_B, J^0_B, J^1_B, O^0_B, O^1_B\)

\[
B = \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 0 & 1 \\ 1 & 1 \end{bmatrix} : C^{00}_B = \text{Encrypt}(I^0_B \| J^0_B, O^0_B) \quad C^{01}_B = \text{Encrypt}(I^0_B \| J^1_B, O^0_B) \\
C^{10}_B = \text{Encrypt}(I^1_B \| J^0_B, O^1_B) \quad C^{11}_B = \text{Encrypt}(I^1_B \| J^1_B, O^1_B)
\]
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 \( I_{x_1} \)
- for \( x_2 \), she sends \( J_{x_2} \)
- for \( x_3 \), she sends \( J_{x_3} \)

Bob’s Inputs

\[ A = [1 \ 0] \]

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

Oblivious Transfer

Alice owns \( I_A^0, I_A^1 \) and Bob owns \( y_1 \in \{0, 1\} \)
- Using an OT, Bob gets \( I_A^{y_1} \), while Alice learns nothing
- From the ciphertexts \( (C_B^b)_b \), Bob gets \( O_A^{y_1} \)

Bob’s Inputs

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

\[ C_B^{00} = \text{Encrypt}(I_B^0 || J_B^0, O_B^0) \]
\[ C_B^{01} = \text{Encrypt}(I_B^0 || J_B^1, O_B^0) \]
\[ C_B^{10} = \text{Encrypt}(I_B^1 || J_B^0, O_B^0) \]
\[ C_B^{11} = \text{Encrypt}(I_B^1 || J_B^1, O_B^0) \]

Oblivious Transfer

Alice owns \( I_B^0, I_B^1 \) and Bob owns \( y_2 \in \{0, 1\} \)
- Using an OT, Bob gets \( I_B^{y_2} \), while Alice learns nothing
- Bob additionally knows \( J_B^{y_2} \)
- From the ciphertexts \( (C_B^{bb'})_{bb'} \), Bob gets \( O_B^{y_2} \)

Internal Garbled Gates

Internal Garbled Gate

For the gate \( E \) (or): 2 new random secret keys \( O_E^0, O_E^1 \) while \( I_E^0 \leftarrow O_A^0, I_E^1 \leftarrow O_A^1, J_E^0 \leftarrow O_B^0, J_E^1 \leftarrow O_B^1 \)

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

\[ C_E^{00} = \text{Encrypt}(I_E^0 || J_E^0, O_E^0) \]
\[ C_E^{01} = \text{Encrypt}(I_E^0 || J_E^1, O_E^1) \]
\[ C_E^{10} = \text{Encrypt}(I_E^1 || J_E^0, O_E^1) \]
\[ C_E^{11} = \text{Encrypt}(I_E^1 || J_E^1, O_E^1) \]
Evaluation of Internal Gates

\[
E = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}: C_E^{00} = \text{Encrypt}(I_E^0 || J_E^0, O_E^0) \quad C_E^{01} = \text{Encrypt}(I_E^0 || J_E^1, O_E^1) \\
C_E^{10} = \text{Encrypt}(I_E^1 || J_E^0, O_E^1) \quad C_E^{11} = \text{Encrypt}(I_E^1 || J_E^1, O_E^1)
\]

Bob knows \(I_E^A = O_E^A\) and \(J_E^B = O_E^B\)
From the ciphertexts \((C_{bb'}_{E})_{bb'}\), Bob gets \(O_E^{Y_E}\)

Evaluation of Gate G

\[
G = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}: C_G^{00} = \text{Encrypt}(I_G^0 || J_G^0, 0) \quad C_G^{01} = \text{Encrypt}(I_G^0 || J_G^1, 1) \\
C_G^{10} = \text{Encrypt}(I_G^1 || J_G^0, 1) \quad C_G^{11} = \text{Encrypt}(I_G^1 || J_G^1, 1)
\]

Bob knows \(I_G^E = O_G^E\) and \(J_G^F = O_G^F\)
From the ciphertexts \((C_{bb'}_{G})_{bb'}\), 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
Honest-but-Curious and Malicious

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