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 $y_i = f_i(x_1, \ldots, x_n)$
- \ldots 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 $x$ and Bob owns a value $y$, in the end, they both learn whether $x = y$ or not

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

Equal 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 $C' = E(r(x - y))$, for a random element $r$
- Alice computes $C'' = E(r'(x - y))$, for a random element $r'$
- They jointly decrypt $C''$: the value is 0 iff $x = y$ (or random)

Yao Millionaires’ Problem

We fill and order the sets by length: $\bar{T}_1^x = \{X_i\}$ and $\bar{T}_0^y = \{Y_i\}$ where

- for $i = 0, \ldots, n$:
  - if $x_i = 0$, $X_i = 2^n$, otherwise $X_i = x_{n-1} \ldots x_i \in [0, 2^{n-i}]$
  - if $y_i = 1$, $Y_i = 2^n + 1$, otherwise $Y_i = y_{n-1} \ldots y_i + 1 \in [0, 2^{n-i}]$

$x > y \iff \exists i < n, X_i = Y_i$

With Homomorphic Encryption

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

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

2. Oblivious Transfer

3. Garbled Circuits

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 $g((x_0, x_1), b) = \bot$.

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

Example (Bellare-Micali’s Construction – 1992)
In a discrete logarithm setting \((G, g, p)\), for \(x_0, x_1 \in G\)

- Alice chooses \(c \leftarrow R G\) and sends it to Bob
- Bob chooses \(k \leftarrow Z_p\), 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:
  - \(C_i \leftarrow g^{r_i}\) and \(C'_i \leftarrow x_i \cdot pk_i^{r_i}\), for \(r_i \leftarrow Z_p\)
- 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 \(r, s, t \leftarrow Z_p\), 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:
  - \(T_0 \leftarrow (X, Y_0' = Y^{u_0}g^{v_0}, Z_0' = Z_0^{u_0}X^{v_0})\)
  - \(T_1 \leftarrow (X, Y_1' = Y^{u_1}g^{v_1}, Z_1' = Z_1^{u_1}X^{v_1}), for u_0, v_0, u_1, v_1 \leftarrow Z_p\)
- Alice encrypts \(x_i\) under \(T_j\): \(C_i = Y_i'\) 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

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.

Outline

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

Garbled Circuit

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

1-Input Garbled Gate
For the gate A (not): 4 random secret keys \(I_0^A, I_1^A, O_0^A, O_1^A\)

\[
A = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad \text{not} \quad \text{and} \quad C_0^A = \text{Encrypt}(I_0^A, O_1^A) \quad C_1^A = \text{Encrypt}(I_1^A, O_0^A)
\]

2-Input Garbled Gate
For the gate B (and): 8 random secret keys \(I_0^B, I_1^B, J_0^B, J_1^B, O_0^B, O_1^B\)

\[
B = \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 1 \\ 1 & 1 \end{bmatrix} 
\begin{array}{c}
\text{and} \\
\text{or} \\
\text{line} \\
\text{or}
\end{array} 
C_{00}^B = \text{Encrypt}(I_0^B||J_0^B, O_0^B) \quad C_{01}^B = \text{Encrypt}(I_0^B||J_1^B, O_0^B) \\
C_{10}^B = \text{Encrypt}(I_1^B||J_0^B, O_0^B) \quad C_{11}^B = \text{Encrypt}(I_1^B||J_1^B, O_0^B)
\]

Garbled Gates

Alice generates the garbled gates
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

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

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 \((C_A^b)_b\), Bob gets \( O_A^{y_A} \)

Internal Garbled Gates

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

Oblivious Transfer

Alice owns \( I_A^0, I_A^1 \) and Bob owns \( y_2 \in \{0, 1\} \)
- Using an OT, Bob gets \( I_{A}^{y_2} \), while Alice learns nothing
- Bob additionally knows \( J_{x_2} \)
- From the ciphertexts \((C_B^{bb'})_{bb'}\), Bob gets \( O_B^{y_B} \)
**Evaluation of Internal Gates**

\[
E = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix} : 
C_{00}^E = \text{Encrypt}(I_0^E || J_0^E, O_0^E) \\
C_{01}^E = \text{Encrypt}(I_0^E || J_1^E, O_0^E) \\
C_{10}^E = \text{Encrypt}(I_1^E || J_0^E, O_1^E) \\
C_{11}^E = \text{Encrypt}(I_1^E || J_1^E, O_1^E)
\]

**Evaluation of Gate E**

Bob knows \( I_A^E = O_A^E \) and \( J_B^E = O_B^E \)

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

---

**Output Garbled Gates**

\[
G = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix} : 
C_{00}^G = \text{Encrypt}(I_0^G || J_0^G, 0) \\
C_{01}^G = \text{Encrypt}(I_0^G || J_1^G, 1) \\
C_{10}^G = \text{Encrypt}(I_1^G || J_0^G, 1) \\
C_{11}^G = \text{Encrypt}(I_1^G || J_1^G, 1)
\]

**Evaluation of Gate G**

Bob knows \( I_A^G = O_A^G \) and \( J_B^G = O_B^G \)

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
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