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

\textit{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}

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

Outline

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

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

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

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

\[
T_{x}^{1} = \{x_{n-1} \ldots x_{i}|x_{i} = 1\} \quad T_{y}^{0} = \{y_{n-1} \ldots y_{i+1}|y_{i} = 0\}
\]

\[x > y \iff T_{x}^{1} \cap T_{y}^{0} \neq \emptyset\]

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 \( C' = E(r(x - y)) \), 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

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

We fill and order the sets by length: \( \bar{T}_{x}^{1} = \{X_{i}\} \) and \( \bar{T}_{y}^{0} = \{Y_{i}\} \) where for \( i = 0, \ldots, n \):

- if \( x_{i} = 0 \), \( X_{i} = 2^{n} \), otherwise \( X_{i} = x_{n-1} \ldots x_{i} \in [0, 2^{n-1}[ \)
- if \( y_{i} = 1 \), \( Y_{i} = 2^{n} + 1 \), otherwise \( Y_{i} = y_{n-1} \ldots y_{i+1}1 \in [0, 2^{n-1}[ \)

\[x > y \iff \exists i < n, X_{i} = Y_{i}\]

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 \( x > y \)
Secure Function Evaluation

Introduction
Examples
Malicious Setting

Oblivious Transfer

Definition
Examples

Garbled Circuits

Outline

GMW Compiler

[Goldreich-Micali-Wigderson – STOC 1987]

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

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

[Rabin – 1981]

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 $g((x_0, x_1), b) = \perp$

[Kilian – STOC 1988]

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 (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:
  \(T_0 \leftarrow (X, Y'_0 = Y^{u_0} g^{v_0}, Z'_0 = Z_0^{u_0} X^{v_0})\) and
  \(T_1 \leftarrow (X, Y'_1 = Y^{u_1} g^{v_1}, Z'_1 = Z_1^{u_1} X^{v_1})\), for \(u_0, v_0, u_1, v_1 \overset{R}{\leftarrow} \mathbb{Z}_p\).
- Alice encrypts \(x_i\) under \(T_i\): \(C_i = Y^i_i\) 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

Garbled Circuit

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

\[
A = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad \text{not}
\]

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

\[
C = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix} \quad \text{or}
\]

\[
D = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \quad \text{line}
\]

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

\[
F = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix} \quad \text{and}
\]

\[
G = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix} \quad \text{or}
\]

Garbled Gates

Alice generates the garbled gates

1-Input Garbled Gate

For the gate A (not): 4 random secret keys \(I_A^0, I_A^1, O_A^0, O_A^1\)

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

2-Input Garbled Gate

For the gate B (and): 8 random secret keys \(I_B^0, I_B^1, J_B^0, J_B^1, O_B^0, O_B^1\)

\[
B = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} : C_B^{00} = \text{Encrypt}(I_B^0 || J_B^0, O_B^0) \quad C_B^{01} = \text{Encrypt}(I_B^0 || J_B^1, O_B^0) \quad C_B^{10} = \text{Encrypt}(I_B^1 || J_B^0, O_B^0) \quad C_B^{11} = \text{Encrypt}(I_B^1 || J_B^1, O_B^0)
\]
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 \( I_{D_1} \)
- for \( x_2 \), she sends \( J_{B_2} \)
- for \( x_3 \), she sends \( J_{C_3} \)

Bob’s Inputs

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

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 \( O_{A_y}^b \)

Bob’s Inputs

\[
B = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}: C_B^{00} = \text{Encrypt}(I_B^0 || J_B^0, O_B^0) \quad C_B^{01} = \text{Encrypt}(I_B^0 || J_B^1, O_B^0)
\]

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

Oblivious Transfer

Alice owns \( I_B^0, I_B^1 \), and Bob owns \( y_2 \in \{0, 1\} \)

- Using an OT, Bob gets \( I_{B_y^0} \), while Alice learns nothing
- Bob additionally knows \( J_{B_y^0} \)
- From the ciphertexts \( (C_B^b)_b \), Bob gets \( O_{B_y}^b \)

Internal Garbled Gates

For the gate \( E \) (or): 2 new random secret keys \( O_{E_0}^0, O_{E_1}^1 \) while \( I_{E_0}^0 \leftarrow O_A^0, I_{E_1}^1 \leftarrow O_A^1, J_{E_0}^0 \leftarrow O_B^0, J_{E_1}^1 \leftarrow O_B^1 \)

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

\[
C_E^{10} = \text{Encrypt}(I_E^1 || J_E^0, O_E^1) \quad C_E^{11} = \text{Encrypt}(I_E^1 || J_E^1, O_E^1)
\]

Internal Garbled Gate

- For the gate \( E \) (or): 2 new random secret keys \( O_{E_0}^0, O_{E_1}^1 \) while \( I_{E_0}^0 \leftarrow O_A^0, I_{E_1}^1 \leftarrow O_A^1, J_{E_0}^0 \leftarrow O_B^0, J_{E_1}^1 \leftarrow O_B^1 \)
- Bob additionally knows \( J_{E_y^0} \)
- From the ciphertexts \( (C_B^b)_b \), Bob gets \( O_{B_y}^b \)
Evaluation of Internal Gates

A
B
y1 yA
x2
y2 yB
yE
E =
\[
\begin{bmatrix}
0 & 1 \\
1 & 1
\end{bmatrix}
\]

Evaluation of Gate E

Bob knows \( \mathcal{I}^{\mathcal{Y}_E}_A = \mathcal{O}^{\mathcal{Y}_E}_A \) and \( \mathcal{J}^{\mathcal{Y}_E}_B = \mathcal{O}^{\mathcal{Y}_E}_B \)

From the ciphertexts \( (\mathcal{C}_{bb'} \mathcal{G})_{bb'} \), Bob gets \( \mathcal{O}^{\mathcal{Y}_E}_E \)

Evaluation of Internal Gates

For the gate \( \mathcal{G} \) (or):

Bob knows \( \mathcal{I}^{\mathcal{Y}_E}_G = \mathcal{O}^{\mathcal{Y}_E}_G \) and \( \mathcal{J}^{\mathcal{Y}_F}_G = \mathcal{O}^{\mathcal{Y}_F}_G \)

From the ciphertexts \( (\mathcal{C}_{bb'} \mathcal{G})_{bb'} \), Bob gets \( \mathcal{O}^{\mathcal{Y}_E}_E \)

Bob can then transmit \( \mathcal{Z} \) 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
  \(\Rightarrow\) Redundancy is added to the plaintext
  (or authenticated encryption)

They have to trust each other

- Alice correctly builds garbled gates: the ciphertexts are correct
  \(\Rightarrow\) Cut-and-choose technique

- Alice plays the oblivious transfer protocols with correct inputs
  \(\Rightarrow\) Inputs are committed, checked during the cut-and-choose,
  and ZK proofs are done during the OT

- Bob sends back the correct value \(z\)
  \(\Rightarrow\) Random tags are appended to the final results 0 and 1
  that Bob cannot guess