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.

Electronics 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

Apple owns an integer $x$ and Bob owns an integer $y$, in the end, they both learn whether $x \leq y$ 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 $C^\oplus = E(r^0(x - y))$, for a random element $r^0$.
- They jointly decrypt $C^\oplus$; 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

- if $x_i = 0$, $X_i = 2^n - 1$, otherwise $X_i = x_{n-1} \ldots x_i$.
- if $y_i = 1$, $Y_i = 2^n + 1$, otherwise $Y_i = y_{n-1} \ldots y_i + 1$.

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

With Homomorphic Encryption

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

\[x > y \iff \exists i < n, (x_i > y_i) \wedge (\forall j > i, x_j = y_j)\]
\[\iff \exists i < n, (x_i = 1) \wedge (y_i = 0) \wedge (\forall j > i, x_j = y_j)\]
\[\iff \exists i < n, (y_i = 0) \wedge (x_{n-1} \ldots x_i = y_{n-1} \ldots y_{i+1})\]
\[\iff |T_x^1 \cap T_y^0| = 0\]
Outline

1 Secure Function Evaluation
   - Introduction
   - Examples
   - Malicious Setting

2 Oblivious Transfer

3 Garbled Circuits

GMW Compiler

[Goldreich-Micali-Wigderson – STOC 1987]

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

Secure Function Evaluation

1 Secure Function Evaluation
   - Introduction
   - Examples
   - Malicious Setting

2 Oblivious Transfer

3 Garbled Circuits

GMW Compiler

[Goldreich-Micali-Wigderson – STOC 1987]

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

Oblivious Transfer

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

Oblivious Transfer is equivalent to Secure 2-Party Computation

[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
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 \(pk_0 \cdot pk_1 \overset{R}{=} c\) and encrypts \(x_i\) under \(pk_i\) (for \(i = 0, 1\)) with ElGamal:
  \[ C_i \leftarrow g^{r_i} \text{ and } C_i^0 \leftarrow x_i \cdot pk_i^{r_i}, \text{ for } r_i \overset{R}{\leftarrow} \mathbb{Z}_p \]
- 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 \(r, s, t \overset{R}{\leftarrow} \mathbb{Z}_p\), 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:
  \[ T_0 \leftarrow (X, Y_0^0 = Y^u_0 g^{v_0}, Z_0^0 = Z_0^{u_0} X^{v_0}) \text{ and } \]
  \[ T_1 \leftarrow (X, Y_1^0 = Y^u_1 g^{v_1}, Z_1^0 = Z_1^{u_1} X^{v_1}) \text{, for } u_0, v_0, u_1, v_1 \overset{R}{\leftarrow} \mathbb{Z}_p \]
- Alice encrypts \(x_i\) under \(T_i\): \(C_i = Y_i^0\) 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, 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** (not):
  - \(A = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}\):
  - \(C_A^0 = \text{Encrypt}(I_A^0, O_A^0)\)
  - \(C_A^1 = \text{Encrypt}(I_A^1, O_A^1)\)

- **B** (and):
  - \(B = \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 0 & 1 \\ 1 & 1 \end{bmatrix}\):
  - \(C_B^{00} = \text{Encrypt}(I_B^0, J_B^0, O_B^0)\)
  - \(C_B^{01} = \text{Encrypt}(I_B^0, J_B^1, O_B^0)\)
  - \(C_B^{10} = \text{Encrypt}(I_B^1, J_B^0, O_B^0)\)
  - \(C_B^{11} = \text{Encrypt}(I_B^1, J_B^1, O_B^0)\)

---

### Garbled Gates

Alice generates the garbled gates

#### 1-Input Garbled Gate

For the gate A (not):
- 4 random secret keys \(I_A^0, I_A^1, O_A^0, O_A^1\)
- \(A = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}\):
  - \(C_A^0 = \text{Encrypt}(I_A^0, O_A^0)\)
  - \(C_A^1 = \text{Encrypt}(I_A^1, O_A^1)\)

#### 2-Input Garbled Gate

For the gate B (and):
- 8 random secret keys \(I_B^0, I_B^1, J_B^0, J_B^1, O_B^0, O_B^1\)
- \(B = \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 0 & 1 \\ 1 & 1 \end{bmatrix}\):
  - \(C_B^{00} = \text{Encrypt}(I_B^0, J_B^0, O_B^0)\)
  - \(C_B^{01} = \text{Encrypt}(I_B^0, J_B^1, O_B^0)\)
  - \(C_B^{10} = \text{Encrypt}(I_B^1, J_B^0, O_B^0)\)
  - \(C_B^{11} = \text{Encrypt}(I_B^1, J_B^1, O_B^0)\)
**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 $I^{x_1}_D$
- For $x_2$, she sends $J^{x_2}_B$
- For $x_3$, she sends $J^{x_3}_C$

**Bob’s Inputs**

$A = [1 \ 0]$

$C^0_A = \text{Encrypt}(I^0_A, O^1_A)$

$C^1_A = \text{Encrypt}(I^1_A, O^0_A)$

**Oblivious Transfer**

Alice owns $I^0_A$, $I^1_A$ and Bob owns $y_1 \in \{0, 1\}$

- Using an OT, Bob gets $I^y_A$, while Alice learns nothing
- From the ciphertexts $(C^y_A)$, Bob gets $O^y_A$

**Internal Garbled Gates**

For the gate $E$ (or): 2 new random secret keys $O^0_E$, $O^1_E$

while $I^0_E \leftarrow O^0_A$, $I^1_E \leftarrow O^1_A$, $J^0_E \leftarrow O^0_B$, $J^1_E \leftarrow O^1_B$

$E = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}$

$C^{00}_E = \text{Encrypt}(I^0_E \| J^0_E, O^0_E)$

$C^{01}_E = \text{Encrypt}(I^0_E \| J^1_E, O^1_E)$

$C^{10}_E = \text{Encrypt}(I^1_E \| J^0_E, O^0_E)$

$C^{11}_E = \text{Encrypt}(I^1_E \| J^1_E, O^1_E)$
Evaluation of Internal Gates

\[ \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix} : \]

- \( C_E^{00} = \text{Encrypt}(I_E^0 || J_E^0, O_E^0) \)
- \( C_E^{01} = \text{Encrypt}(I_E^0 || J_E^1, O_E^1) \)
- \( C_E^{10} = \text{Encrypt}(I_E^1 || J_E^0, O_E^1) \)
- \( C_E^{11} = \text{Encrypt}(I_E^1 || J_E^1, O_E^1) \)

Evaluation of Gate E

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

Evaluation of Gate G

Bob knows \( I_G^E = O_G^E \) and \( J_G^F = O_G^F \)
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
Honest-but-Curious and Malicious

The previous construction assumes that

- Bob extracts the correct plaintext among the multiple candidates
  ⇒ Redundancy is added to the plaintext
    (or authenticated encryption)

They have to trust each other

- Alice correctly builds garbled gates: the ciphertexts are correct
  ⇒ Cut-and-choose technique

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

- Bob sends back the correct value $z$
  ⇒ Random tags are appended to the final results 0 and 1
    that Bob cannot guess