Secure Function Evaluation

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

2. Oblivious Transfer
   - Definition
   - Examples

3. Garbled Circuits
   - Introduction
   - Garbled Circuits
   - Correctness

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

A low owns a value $x$ and Bob owns a value $y$,
in the end, they both learn whether $x = y$ or not

Yao Millionaires’ Problem

A low 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]

| $x > y$ | $\exists! i < n, (x_i > y_i)$ \& (\forall j > i, x_j = y_j) \\ $\iff \exists! i < n, (x_i = 1)$ \& (\forall j > i, x_j = y_j) \\ $\iff \exists! i < n, (y_i = 0)$ \& (\forall j > i, x_j = y_j) \\ $\iff |T_X^1 \cap T_Y^0| = 1$ |

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 $r_i$
- 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

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 $x = y$ or not

Equality Test

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)
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 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
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 \(c \leftarrow R G\) and sends it to Bob
- Bob chooses \(k \leftarrow R \mathbb{Z}_p\), 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:
  \[C_i \leftarrow g^{r_i} \text{ and } C_i' \leftarrow x_i \cdot pk_i^{r_i}, \text{ for } r_i \leftarrow R \mathbb{Z}_p\]
- 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 \(r, s, t \leftarrow R \mathbb{Z}_p\), 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:
  \[T_0 \leftarrow (X, Y_0' = Y^{u_0} \cdot g^{v_0}, Z_0' = Z_0^{u_0} \cdot X^{v_0})\]
  \[T_1 \leftarrow (X, Y_1' = Y^{u_1} \cdot g^{v_1}, Z_1' = Z_1^{u_1} \cdot X^{v_1})\]
  for \(u_0, v_0, u_1, v_1 \leftarrow R \mathbb{Z}_p\)
- 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

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

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

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

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

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

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

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

Garbled Gates

Alice generates the garbled gates.

1-Input Garbled Gate

For the gate A (not): 4 random secret keys $I^0_A$, $I^1_A$, $O^0_A$, $O^1_A$

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

2-Input Garbled Gate

For the gate B (and): 8 random secret keys $I^0_B$, $I^1_B$, $J^0_B$, $J^1_B$, $O^0_B$, $O^1_B$

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

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

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 \( C_A^{y} \), Bob gets \( O_A^{y} \)

Bob’s Inputs

\[ B = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} : C_B^{00} = \text{Encrypt}(I_B^0 || J_B^0, O_B^0) \quad C_B^{01} = \text{Encrypt}(I_B^0 || J_B^1, O_B^0) \]
\[ C_B^{10} = \text{Encrypt}(I_B^1 || J_B^0, O_B^1) \quad C_B^{11} = \text{Encrypt}(I_B^1 || J_B^1, O_B^1) \]

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 \( C_B^{y_2} \), Bob gets \( O_B^{y_2} \)

Internal Garbled Gates

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

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

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

- \( C_{E}^{00} = \text{Encrypt}(I_{E}^{0} \parallel J_{E}^{0}, O_{E}^{0}) \)
- \( C_{E}^{01} = \text{Encrypt}(I_{E}^{0} \parallel J_{E}^{1}, O_{E}^{1}) \)
- \( C_{E}^{10} = \text{Encrypt}(I_{E}^{1} \parallel J_{E}^{0}, O_{E}^{1}) \)
- \( C_{E}^{11} = \text{Encrypt}(I_{E}^{1} \parallel J_{E}^{1}, O_{E}^{1}) \)

**Evaluation of Gate E**

Bob knows \( I_{E}^{yA} = O_{A}^{yA} \) and \( J_{E}^{yB} = O_{B}^{yB} \)

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

**Output Garbled Gates**

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

- \( C_{G}^{00} = \text{Encrypt}(I_{G}^{0} \parallel J_{G}^{0}, 0) \)
- \( C_{G}^{01} = \text{Encrypt}(I_{G}^{0} \parallel J_{G}^{1}, 1) \)
- \( C_{G}^{10} = \text{Encrypt}(I_{G}^{1} \parallel J_{G}^{0}, 1) \)
- \( C_{G}^{11} = \text{Encrypt}(I_{G}^{1} \parallel J_{G}^{1}, 1) \)

**Evaluation of Gate G**

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

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

---

ENS/CNRS/INRIA Cascade
David Pointcheval
29/33

ENS/CNRS/INRIA Cascade
David Pointcheval
30/33

ENS/CNRS/INRIA Cascade
David Pointcheval
31/33

ENS/CNRS/INRIA Cascade
David Pointcheval
32/33
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 $z$
  \[ \Rightarrow \text{Random tags are appended to the final results 0 and 1 that Bob cannot guess} \]