IV - Secure Function Evaluation and Secure 2-Party Computation

David Pointcheval
Ecole normale supérieure, CNRS & INRIA

Outline

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

2. Oblivious Transfer
   - Definition
   - Examples

3. Garbled Circuits
   - Introduction
   - Garbled Circuits
   - Correctness

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*ₗ’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 ∈ {0, 1} and *f*(*x*₁, ..., *x*ₙ) = ∑ *j* *x*ᵢ

**Example (Homomorphic Encryption)**
- *P*ᵢ encrypts *C*ᵢ = *E*(*x*ᵢ) with an additively homomorphic encryption scheme
- They all compute *C* = *E*(∑ *x*ᵢ)
- They jointly decrypt *C* to get *y* = ∑ *x*ᵢ using a distributed decryption

Electronic Voting

**Privacy: Limitations**
In case of unanimity (i.e. ∑ *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

$T^1_x = \{x_{n-1} \ldots x_i | x_i = 1\}$

$T^0_y = \{y_{n-1} \ldots y_i+1 | y_i = 0\}$

$x > y \iff T^1_x \cap T^0_y \neq \emptyset$

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

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 $Y_i = y_{n-1} \ldots y_i+1 \in [0, 2^{n-i}[$

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

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 Function Evaluation

1. Introduction
2. Examples
3. Malicious Setting

Oblivious Transfer

Definition

Examples

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

Able 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 \(c \leftarrow G\) 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:
  \[C_i \leftarrow g^r \text{ and } C'_i \leftarrow x_i \cdot pk_i^r, \text{ for } r \leftarrow \mathbb{Z}_p\]
- 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 \(r, s, t \leftarrow \mathbb{Z}_p\), 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:
  \[T_0 \leftarrow (X, Y'_0 = Y^u_0 g^{v_0}, Z'_0 = Z^u_0 X^{v_0})\] and \[T_1 \leftarrow (X, Y'_1 = Y^u_1 g^{v_1}, Z'_1 = Z^u_1 X^{v_1})\], for \(u_0, v_0, u_1, v_1 \leftarrow \mathbb{Z}_p\).
- Alice encrypts \(x_i\) under \(T_i\): \(C_i = Y'_i\) 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, 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:

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

Garbled Gates

Alice generates the garbled gates:

1-Input Garbled Gate

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

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

2-Input Garbled Gate

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

\[
B = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} : C_B^{00} = \text{Encrypt}(I_B^0 || J_B^0, O_B^0) \quad C_B^{01} = \text{Encrypt}(I_B^0 || J_B^1, O_B^0) \\
C_B^{10} = \text{Encrypt}(I_B^1 || J_B^0, O_B^0) \quad C_B^{11} = \text{Encrypt}(I_B^1 || J_B^1, O_B^0)
\]
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 $I_{D}^{x_1}$
- For $x_2$, she sends $J_{B}^{x_2}$
- For $x_3$, she sends $J_{C}^{x_3}$

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 $(C_{B}^{y_0})_{B}$, Bob gets $O_{A}^{y_A}$

Internal Garbled Gates

\[ B = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} : C_{B}^{00} = \text{Encrypt}(I_{B}^{0} || J_{B}^{0}, O_{B}^{0}) \quad C_{B}^{01} = \text{Encrypt}(I_{B}^{0} || J_{B}^{1}, O_{B}^{0}) \]
\[ C_{B}^{10} = \text{Encrypt}(I_{B}^{1} || J_{B}^{0}, O_{B}^{1}) \quad C_{B}^{11} = \text{Encrypt}(I_{B}^{1} || J_{B}^{1}, O_{B}^{1}) \]

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 $(C_{B}^{bb})_{bb}$, Bob gets $O_{B}^{y_B}$

Internal Garbled Gate

For the gate $E$ (or): 2 new random secret keys $O_{E}^{0}, O_{E}^{1}$ while $I_{E}^{0} \leftarrow O_{A}^{0}, I_{E}^{1} \leftarrow O_{A}^{1}, J_{E}^{0} \leftarrow O_{B}^{0}, J_{E}^{1} \leftarrow O_{B}^{1}$

\[ E = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix} : C_{E}^{00} = \text{Encrypt}(I_{E}^{0} || J_{E}^{0}, O_{E}^{0}) \quad C_{E}^{01} = \text{Encrypt}(I_{E}^{0} || J_{E}^{1}, O_{E}^{1}) \]
\[ C_{E}^{10} = \text{Encrypt}(I_{E}^{1} || J_{E}^{0}, O_{E}^{1}) \quad C_{E}^{11} = \text{Encrypt}(I_{E}^{1} || J_{E}^{1}, O_{E}^{1}) \]
Evaluation of Internal Gates

\[
E = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix} : \begin{align*}
C^0_E &= \text{Encrypt}(I_0^E || J_0^E, O_0^E) \\
C^0_1 &= \text{Encrypt}(I_1^E || J_1^E, O_1^E) \\
C^{10}_E &= \text{Encrypt}(I_1^E || J_0^E, O_1^E) \\
C^{11}_E &= \text{Encrypt}(I_1^E || J_1^E, O_1^E)
\end{align*}
\]

**Evaluation of Gate E**

Bob knows \(I^E_A = O^A_Y\) and \(J^E_B = O^B_Y\)
From the ciphertexts \((C^{bb}_{bb})_{b \neq b}\), Bob gets \(O^Y_E\)

---

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
  \( \Rightarrow \) 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