IV – Secure Function Evaluation and Secure 2-Party Computation

David Pointcheval

Ecole normale supérieure, CNRS & INRIA

ENS – Paris – 2018

Outline

Outline

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

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

Outline

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

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 $C'' = E(rr'(x - y))$, 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

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

$$T_x^1 = \{x_{n-1} \ldots x_i | x_i = 1\} \qquad T_y^0 = \{y_{n-1} \ldots y_{i+1} 1 | y_i = 0\}$$

$$x > y \iff T_x^1 \cap T_y^0 \neq \emptyset$$

$$\begin{aligned} x > y &\iff \exists! i < n, (x_i > y_i) \wedge (\forall j > i, x_j = y_j) \\ &\iff \exists! i < n, (x_i = 1) \wedge (y_i = 0) \wedge (\forall j > i, x_j = y_j) \\ &\iff \exists! i < n, (y_i = 0) \wedge (x_{n-1} \ldots x_i = y_{n-1} \ldots y_{i+1} 1) \\ &\iff |T_x^1 \cap T_y^0| = 1 \end{aligned}$$

Yao Millionaires' Problem

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

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

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 $C_i'' = E(r_i r_i'(X_i - Y_i))$, 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

GMW Compiler

GMW Compiler [Goldreich-Micali-Wigderson – STOC 1987]

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

Outline

1. Secure Function Evaluation
2. **Oblivious Transfer**
   - Definition
   - Examples
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 [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

Outline

Oblivious Transfer

Example (Bellare-Micali's Construction – 1992)

In a discrete logarithm setting $(\mathbb{G}, g, p)$, for $x_0, x_1 \in \mathbb{G}$

- Alice chooses $c \stackrel{R}{\leftarrow} \mathbb{G}$ and sends it to Bob
- Bob chooses $k \stackrel{R}{\leftarrow} \mathbb{Z}_p$, sets $pk_b \leftarrow g^k$ and $pk_{1-b} \leftarrow c/pk_b$, and sends (pk_0, pk_1) to Alice
- Alice checks $pk_0 \cdot pk_1 = c$ 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 $r_i \stackrel{R}{\leftarrow} \mathbb{Z}_p$
- 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**

Oblivious Transfer

Example (Naor-Pinkas Construction – 2000)

In a discrete logarithm setting $(\mathbb{G}, g, p)$, for $x_0, x_1 \in \mathbb{G}$

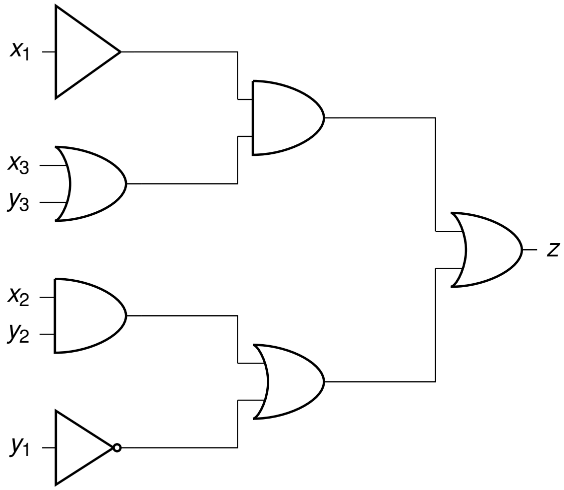
- Bob chooses $r, s, t \stackrel{R}{\leftarrow} \mathbb{Z}_p$, sets $X \leftarrow g^r$, $Y \leftarrow g^s$, $Z_b \leftarrow g^{rs}$, $Z_{1-b} \leftarrow g^t$, and sends (X, Y, Z_0, Z_1) to Bob
- Alice checks $Z_0 \neq Z_1$, and re-randomizes the tuples:
  $T_0 \leftarrow (X, Y'_0 = Y^{u_0} g^{v_0}, Z'_0 = Z_0^{u_0} X^{v_0})$ and
  $T_1 \leftarrow (X, Y'_1 = Y^{u_1} g^{v_1}, Z'_1 = Z_1^{u_1} X^{v_1})$, for $u_0, v_0, u_1, v_1 \stackrel{R}{\leftarrow} \mathbb{Z}_p$
- 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**

Outline

Boolean Circuit

Boolean circuit, Alice's inputs (x_1, x_2, x_3), and Bob's inputs (y_1, y_2, y_3):

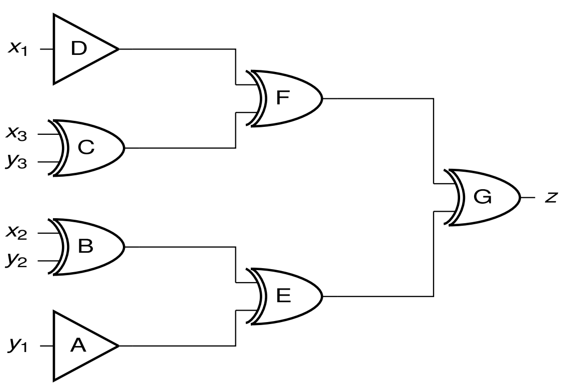


They both learn z in the end, but nothing else

Outline

Garbled Circuit

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



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

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

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

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

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

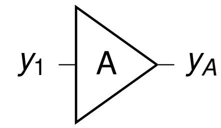
Alice's Inputs

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



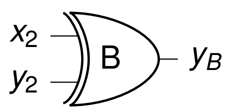
A = $\begin{bmatrix}1 & 0\end{bmatrix}$: $C_A^0 = \text{Encrypt}(I_A^0, O_A^1) \quad C_A^1 = \text{Encrypt}(I_A^1, O_A^0)$

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

Bob's Inputs



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

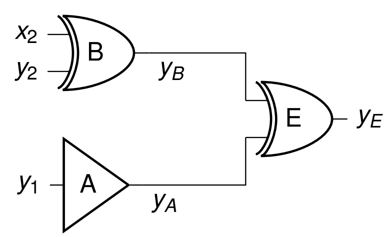
$C_B^{10} = \text{Encrypt}(I_B^1||J_B^0, O_B^0) \quad C_B^{11} = \text{Encrypt}(I_B^1||J_B^1, O_B^1)$

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 $J_B^{x_2}$
- From the ciphertexts $(C_B^{bb'})_{bb'}$, Bob gets $O_B^{y_B}$

Internal Garbled Gates



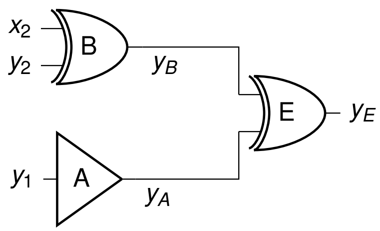
Internal Garbled Gate

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

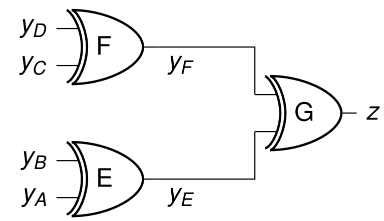


$$E = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix} : \begin{array}{ll} C_E^{00} = \mathrm{Encrypt}(I_E^0 || J_E^0, O_E^0) & C_E^{01} = \mathrm{Encrypt}(I_E^0 || J_E^1, O_E^1) \\ C_E^{10} = \mathrm{Encrypt}(I_E^1 || J_E^0, O_E^1) & C_E^{11} = \mathrm{Encrypt}(I_E^1 || J_E^1, O_E^1) \end{array}$$

Evaluation of Gate E

Bob knows $I_E^{y_A} = O_A^{y_A}$ and $J_E^{y_B} = O_B^{y_B}$
From the ciphertexts $(C_E^{bb'})_{bb'}$, Bob gets $O_E^{y_E}$

Output Garbled Gates

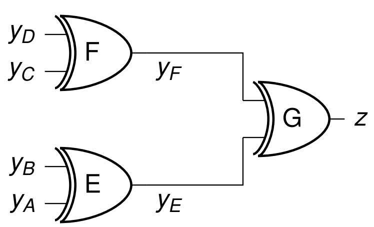


Output Garbled Gate

For the gate G (or): $I_G^0 \leftarrow O_E^0$, $I_G^1 \leftarrow O_E^1$, $J_G^0 \leftarrow O_F^0$, $J_G^1 \leftarrow O_F^1$

$$G = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix} : \begin{array}{ll} C_G^{00} = \mathrm{Encrypt}(I_G^0 || J_G^0, 0) & C_G^{01} = \mathrm{Encrypt}(I_G^0 || J_G^1, 1) \\ C_G^{10} = \mathrm{Encrypt}(I_G^1 || J_G^0, 1) & C_G^{11} = \mathrm{Encrypt}(I_G^1 || J_G^1, 1) \end{array}$$

Evaluation of Internal Gates



$$G = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix} : \begin{array}{ll} C_G^{00} = \mathrm{Encrypt}(I_G^0 || J_G^0, 0) & C_G^{01} = \mathrm{Encrypt}(I_G^0 || J_G^1, 1) \\ C_G^{10} = \mathrm{Encrypt}(I_G^1 || J_G^0, 1) & C_G^{11} = \mathrm{Encrypt}(I_G^1 || J_G^1, 1) \end{array}$$

Evaluation of Gate G

Bob knows $I_G^{y_E} = O_E^{y_E}$ and $J_G^{y_F} = O_F^{y_F}$
From the ciphertexts $(C_G^{bb'})_{bb'}$, Bob gets $z \in \{0, 1\}$
Bob can then transmit z to Alice

Outline

Honest-but-Curious and Malicious

The previous construction assumes that

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

They have to trust each other

- Alice correctly builds garbled gates: the ciphertexts are correct
  $\Longrightarrow$ Cut-and-choose technique
- Alice plays the oblivious transfer protocols with correct inputs
  $\Longrightarrow$ Inputs are committed, checked during the cut-and-choose,
  and ZK proofs are done during the OT
- Bob sends back the correct value z
  $\Longrightarrow$ Random tags are appended to the final results 0 and 1
  that Bob cannot guess