Privacy :

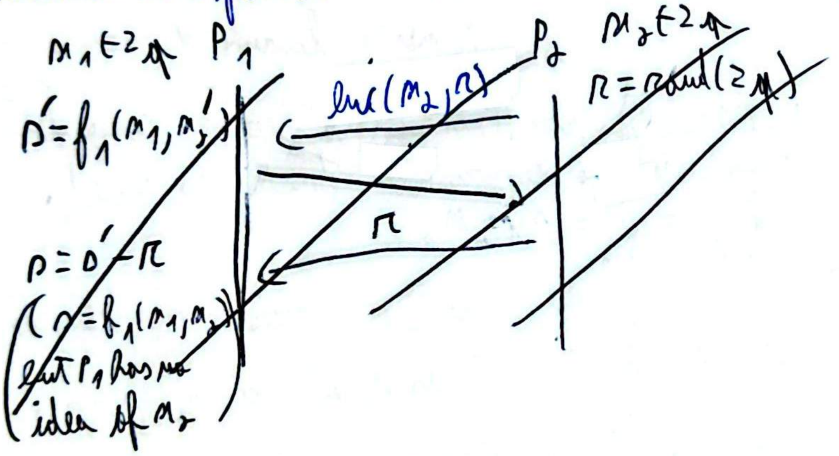
CHAP 5:

order of arguments doesn't matter.

- $\begin{cases} P_i \text{ with input } x_i \\ P_j \text{ with input } x_j \\ \text{ideal outputs } f_i(x_i, x_j) \text{ and } f_j(x_i, x_j) \end{cases}$

- What does a P_i gets when playing Π? => Only what the ideal function allows, nothing more.

- Privacy : Whatever P_i sees during Π must be ~~compatible~~ computable from its own input x_i and/or its output $f_i(x_i, x_j)$ else the simulation fails. $f_1(x_1, x_2) = x_1 + x_2$ $\quad f_2 = \perp$

- 

- Correctness : Even if P_i is malicious it can either only affect the computation by choosing its own input or it cannot force an invalid output. So the result must correspond to some input x_i'.

- Independence : P_i must commit to its input f before learning about anything about x_j. Otherwise, P_i gains strategic advantage.

- No fairness : Secure emulation does not guarantee fairness by default. A malicious party may abort after learning its output or prevent the other from receiving its output. This behavior can be simulated in the ideal world.

EX 0 (5.19):

$f_1(x_1, x_2) = \perp \Rightarrow P_1$ should learn nothing

$f_2(x_1, x_2) = x_1 + x_2 \Rightarrow P_2$ learns the sum

Ideal world P_1 gets $\perp$ and P_2 gets $x_1 + x_2$

P_1 sends x_1 and P_2 learns it and compute f_2.

If P_1 is corrupted ? No danger P_1 learns ~~nothing~~ nothing about P_2.

If P_2 is corrupted ? $\rightarrow$ Real world : $\underline{(x_1, x_2, x_1 + x_2)}$

Ideal world : F takes care to compute functions f_1, f_2

mismatch so no simulator possible. $(x_2, x_1 + x_2)$

$\underline{P_2 \text{ never learns } x_1}$

$\Rightarrow$ A private key is revealed so, Π is not a secure computation as it reveal more informations than allowed.

Ex 1 (5.20):

$f_1 = f_2 = x_1 + x_2$

Corrupted P1 : Real world : Has x_1, receives x_2, computes $x_1 + x_2$
Thus learns $(x_1, x_2, x_1 + x_2)$

Ideal world : Has x_1, receives $f_1 = x_1 + x_2$ from F — mismatch
Thus learns $(x_1, x_1 + x_2)$

No possible simulator

Corrupted P2 : Real world : Has x_2, // ....
Thus learns $(x_1, x_2, x_1 + x_2)$

Ideal world : //
Thus learns $(x_2, x_1 + x_2)$

Same problem

$\Rightarrow$ Π does not securely emulates $F(f_1, f_2)$ because it reveals each party's private key.

Oblivious transfer OT:

~~Sete~~ → Sender P_1 has two secrets m_0, m_1

→ Receiver P_2 has a choice (boolean) $b \in \{0;1\}$

→ Ideal world:

$f_1((m_0, m_1), b) = \perp$ ⇒ Learns nothing

$f_2((m_0, m_1), b) = m_b$ ⇒ learns one message and EXACTLY ONE

P2 gets exactly one message.

P_1 has no idea which one.

P2 learns nothing about the second message.

→ OT captures exactly the kind of asymmetry needed for secure computation. Called complete functionality.

→ Why OT complete? Any function can be represented as a boolean circuit.

→ { Privacy: Only sender learns nothing and receiver only the allowed output.
Independence: Receiver's choice is hidden.
~~No~~ fairness: Sender is allowed to abort early.

→ Based on homomorphic encryption.

P_1		P_2 $b \in \{0;1\}$
$m_0, m_1 \in \mathbb{Z}_p$		$s_b \in \mathbb{Z}_p \wedge r_b \in \mathbb{Z}_p$
$r_0, r_1 = \text{rand}(\mathbb{Z}_p^2)$	$\xleftarrow{r_b,\ \text{Enc}_s(b) = e}$
$_1 =$	

5.3

P_0		P_1
$m_0, m_1 \in \mathbb{Z}_p$		$b \in \{0, 1\}$
$r_0, r_1 = \text{rand}(\mathbb{Z}_p)$	$\xleftarrow{\quad Enc_{pk}(b),\ pk \quad}$	$sk \in \mathbb{Z}_p \wedge pk \in \mathbb{Z}_p$
$c_0 = E_{pk}((1-b)m_0 + b r_0)$	$\xrightarrow{\quad c_0, c_1 \quad}$	If $b = 0$:
$c_1 = E_{pk}(b m_1 + (1-b) r_1)$		$dec_{sk}(c_0) = m_0$
		$dec_{sk}(c_1) = r_1 \rightarrow P_1$ knows it is invalid
$\perp$		else:
		$dec_{sk}(c_0) = r_0 \rightarrow P_1$ knows it is invalid
		$dec_{sk}(c_1) = m_1$

Intuition:

$$E((1-b)m_0 + r_0 b) = E((1-b)m_0) \cdot E(r_0 b) = E(1-b)^{m_0} \cdot E(b)^{r_0}$$

$$= [E(1) \cdot E(-b)]^{m_0} \cdot E(b)^{r_0} = \underbrace{[E(1) \cdot E(b)^{-1}]^{m_0} \cdot E(b)^{r_0} \pmod p}_{\hookrightarrow \text{ Computable}}$$

Secure computation:

- $\rightarrow P_1$ cannot learn b.
- $\rightarrow P_2$ cannot learn m_{1-b} but only m_b.
- $\rightarrow b$ cannot be in function of m_i (b is chosed) before.
- $\rightarrow m_i$ cannot be in function of b (b is encrypted).

Yao's Garbled Circuits:

1) Problem:

- Two parties P_1 and P_2
- They have inputs x_1, x_2
- They want to compute $f(x_1, x_2) = (f_1(x_1, x_2), f_2(x_1, x_2))$
- Privacy goal: Each party only learn their own output.

Honest but curious:

- Only P1 builds the garbled circuit.
- P2 evaluates it using keys received via OT.
- At the end, P2 can compute its output. P1 may get its output back using some "unmasking".

2) Garbling a circuit:

Encode each wire

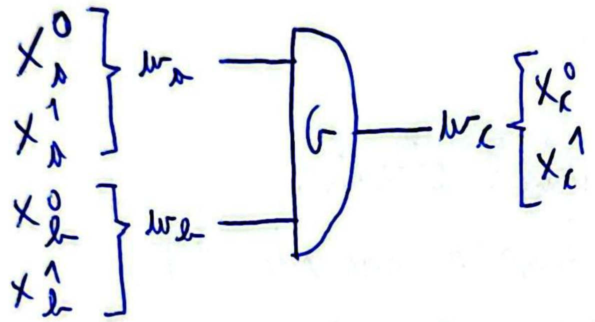
- Encode each wire w in the boolean circuit:
  - Assign two random keys k_w^0, k_w^1
  - These represent 0 or 1 on that wire.
- Output wires: Set keys to actual binary values to allow the final output reading.
- Keys are wire bit values.

Garble each gate

- For a gate g with inputs a, b and output c:
  - Construct a table to decrypt ONLY the correct output key.
  - Table entries: $Enc_{k_a^i}(Enc_{k_b^j}(k_c^{g(i,j)} \| 0^n))$ $(i,j) \in \{(0,0),(0,1),(1,0),(1,1)\}$
  - Shuffle table to make order not leaking anything.
- P2 can decrypt EXACTLY one entry for each gate using its input keys, & doing it with wrong keys give nonsense.
- Privacy of intermediate values.

3) Sending inputs:

- P_1 knows its own keys and send them to P_2.
- P_2 needs its keys without revealing its input:
  - use oblivious transfer OT for each input bit.
  - P2 picks the key corresponding to its input bit.
  - P1 never knows which key P2 choose.



4) Evaluating the circuit:

- P2 has keys for P1's inputs.
- Keys for its own inputs via OT.
- Garbled tables for each gate.
- P2 decrypts tables gate by gate.
- can compute all output wire keys.
- Can read its output (special keys = actual 0/1).