CSCI 1510

- Oblivious Transfer (continued)
- Semi-Honest MPC for Any Function (GMW)
- Malicious MPC (GMW Compiler)
- Program Obfuscation

Feasibility Results

Computational Security:

Semi-honest Oblivious Transfer (OT)

$\Downarrow$

semi-honest MPC for any function with $t < n$ (# corrupted parties)

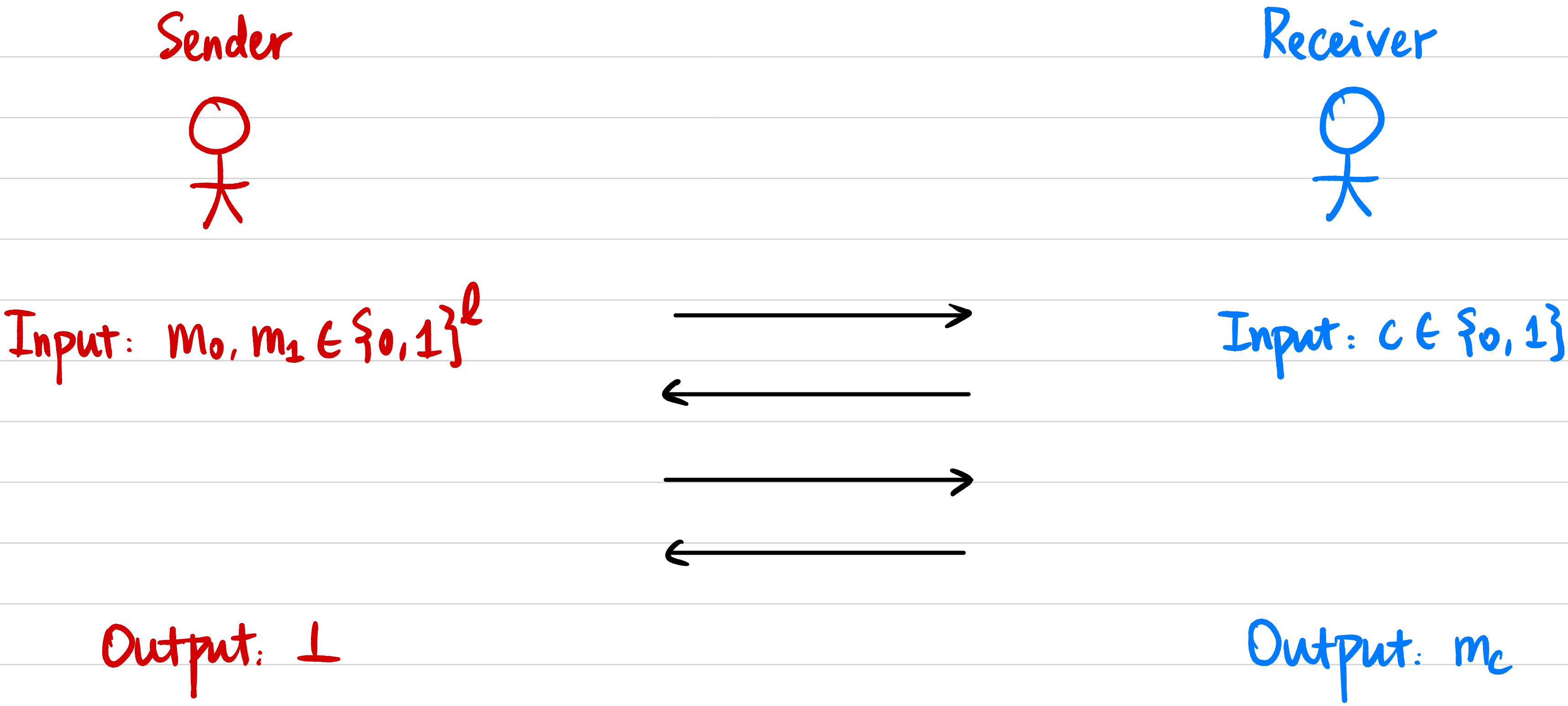
$\Downarrow$

malicious MPC for any function with $t < n$

Information-Theoretic (IT) Security:

semi-honest/malicious MPC for any function with $t < n/2$ (honest majority) ← necessary

Oblivious Transfer (OT)



Oblivious Transfer (OT)

Cyclic group $\mathbb{G}$ of order q with generator g

$H: \mathbb{G} \to \{0,1\}^\ell$

Sender

Input: $m_0, m_1 \in \{0,1\}^\ell$

$a \xleftarrow{\$} \mathbb{Z}_q$

$\xrightarrow{\quad A = g^a \quad}$

Receiver

Input: $c \in \{0,1\}$

$b \xleftarrow{\$} \mathbb{Z}_q$

$\xleftarrow{\quad B = g^b \cdot A^c \quad}$

$k_0 := H(B^a)$

$k_1 := H((\frac{B}{A})^a)$

$\xrightarrow{\quad ct_0 := k_0 \oplus m_0, \; ct_1 := k_1 \oplus m_1 \quad}$

Output: $m_c := ct_c \oplus H(A^b)$

Thm If CDH is hard in $\mathbb{G}$ and H is modeled as a random oracle, then this protocol is semi-honest secure.

$S_A(1^n, (m_0, m_1), \perp)$

Sender

Input: $m_0, m_1 \in \{0,1\}^\ell$

$a \xleftarrow{\$} \mathbb{Z}_q$

$\xrightarrow{\quad A = g^a \quad}$

$\xleftarrow{\quad B \xleftarrow{\$} G \quad}$

$k_0 := H(B^a)$

$k_1 := H\left(\left(\frac{B}{A}\right)^a\right)$

$\xrightarrow[\quad ct_1 := k_1 \oplus m_1 \quad]{\quad ct_0 := k_0 \oplus m_0 \quad}$

$S_A(1^n, c, m_c)$

Receiver

Input: $c \in \{0, 1\}$

$a \xleftarrow{\$} \mathbb{Z}_q$

$\xrightarrow{A = g^a}$

$b \xleftarrow{\$} \mathbb{Z}_q$

$\xleftarrow{B = g^b \cdot A^c}$

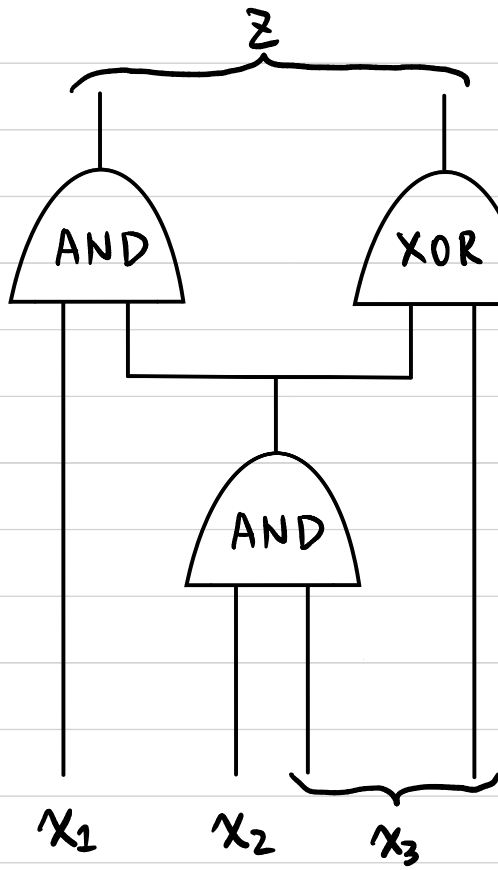
$ct_0 := ?$

$\longrightarrow$

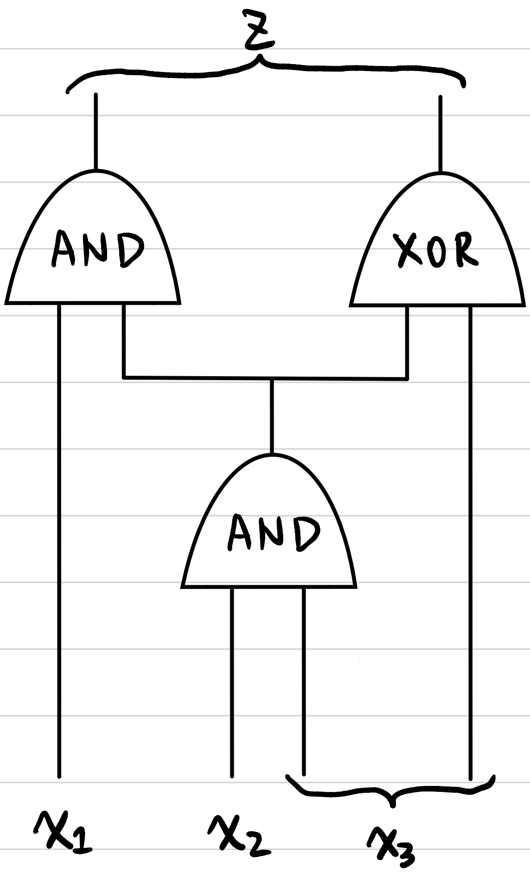
$ct_1 := ?$

Output: $m_c := ct_c \oplus H(A^b)$

Arbitrary Function → Represent it as a Boolean circuit



MPC for any function with $t \le n-1$ (GMW)



Throughout the protocol, we keep the invariant:

For each wire w:

If the value of the wire is $v^w \in \{0,1\}$,

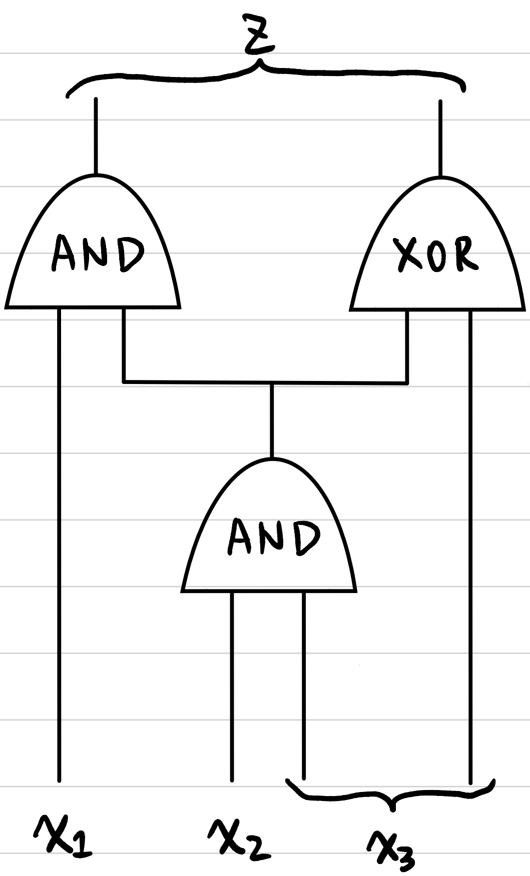
then the n parties hold an additive secret share of v^w

Each party P_i holds a random share $v_i^w \in \{0,1\}$ s.t.

$$\bigoplus_{i=1}^{n} v_i^w = v^w$$

Any $(n-1)$ shares information theoretically hide v^w.

MPC for any function with $t \le n-1$ (GMW)



Each party P_i holds a random share $v_i^w \in \{0,1\}$ s.t. $\bigoplus_{i=1}^{n} v_i^w = v^w$

Inputs:

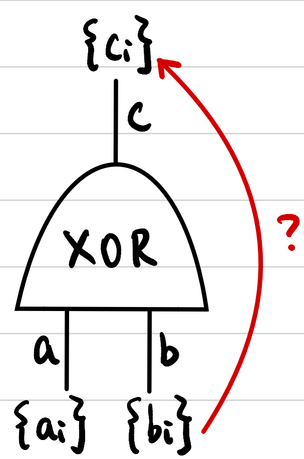
For each input wire w:

If it's from party P_k with input value $v^w \in \{0,1\}$,

P_k randomly samples $v_i^w \xleftarrow{\$} \{0,1\}$ s.t. $\bigoplus_{i=1}^{n} v_i^w = v^w$

→ Sends v_i^w to party P_i.

XOR gates:

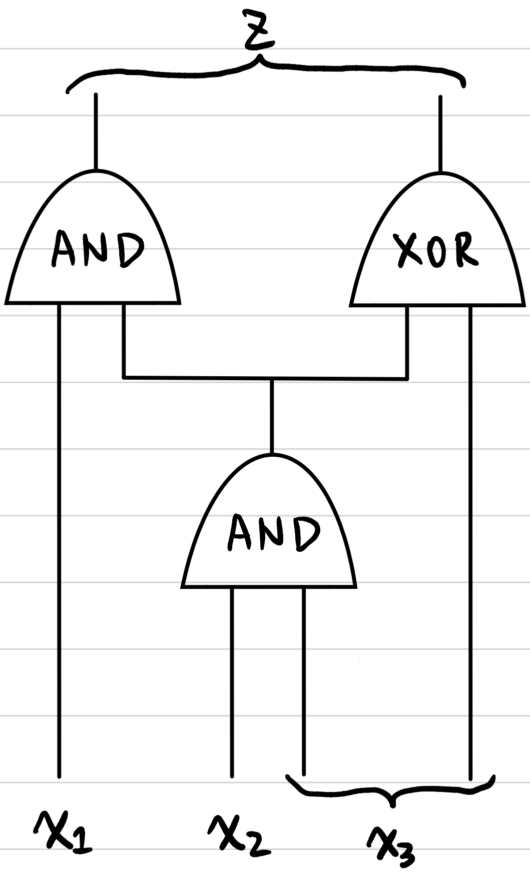


GIVEN: $\bigoplus_{i=1}^{n} a_i = a$ $\bigoplus_{i=1}^{n} b_i = b$

WANT: $\{c_i\}$ s.t. $\bigoplus_{i=1}^{n} c_i = c = a \oplus b$

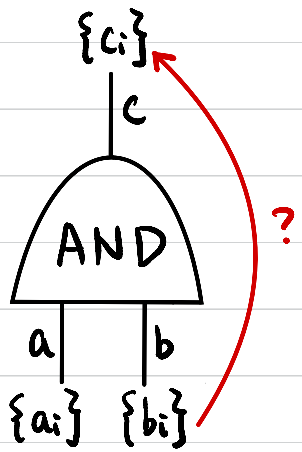
$c_i = ?$

MPC for any function with $t \le n-1$ (GMW)



Each party P_i holds a random share $v_i^w \in \{0,1\}$ s.t. $\bigoplus_{i=1}^{n} v_i^w = v^w$

AND gates:



GIVEN: $\bigoplus_{i=1}^{n} a_i = a$ $\bigoplus_{i=1}^{n} b_i = b$

WANT: $\{c_i\}$ s.t. $\bigoplus_{i=1}^{n} c_i = c = a \cdot b$

$c_i = ?$

Outputs:

For each output wire w:

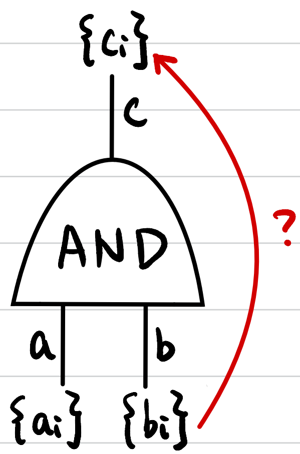
Each party P_i holds a random share $v_i^w \in \{0,1\}$

→ Sends v_i^w to all parties

Each party computes the value $v^w = \bigoplus_{i=1}^{n} v_i^w$

MPC for any function with $t \le n-1$ (GMW)

AND gates:



GIVEN: $\bigoplus_{i=1}^{n} a_i = a$ $\quad \bigoplus_{i=1}^{n} b_i = b$

WANT: $\{c_i\}$ s.t. $\bigoplus_{i=1}^{n} c_i = c = a \cdot b$

$c_i = ?$

$$a \cdot b = \left(\sum_{i=1}^{n} a_i\right) \cdot \left(\sum_{i=1}^{n} b_i\right) \mod 2$$

$$= \left(\sum_{i=1}^{n} a_i \cdot b_i\right) + \left(\sum_{i \neq j} a_i \cdot b_j\right) \mod 2$$

↑ P_i locally $\qquad$ ↑ ?

MPC for any function with $t \le n-1$ (GMW)

Reshare:

P_i | P_j

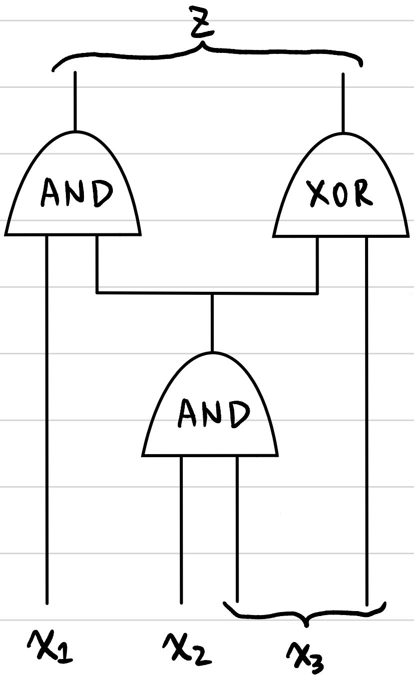
Input: $a_i \in \{0,1\}$ | Input: $b_j \in \{0,1\}$

Output: $r_i \in \{0,1\}$ | Output: $r_j \in \{0,1\}$

WANT: Random $r_i, r_j \in \{0,1\}$ s.t. $r_i + r_j = a_i \cdot b_j \pmod 2$

1) P_i randomly samples $r_i \xleftarrow{\$} \{0,1\}$
2) How to let P_j learn r_j s.t. $r_i + r_j = a_i \cdot b_j \pmod 2$?

MPC for any function with $t \le n-1$ (GMW)



Each party P_i holds a random share $v_i^w \in \{0,1\}$ s.t. $\bigoplus_{i=1}^{n} v_i^w = v^w$

Inputs:

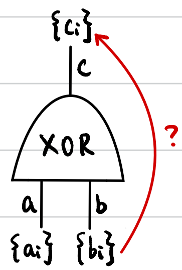
For each input wire w:

If it's from party P_k with input value $v^w \in \{0,1\}$,

P_k randomly samples $v_i^w \xleftarrow{\$} \{0,1\}$ s.t. $\bigoplus_{i=1}^{n} v_i^w = v^w$

↳ Sends v_i^w to party P_i.

XOR gates:

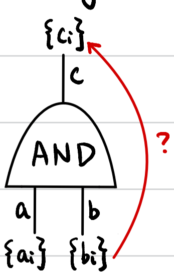


GIVEN: $\bigoplus_{i=1}^{n} a_i = a$ $\quad \bigoplus_{i=1}^{n} b_i = b$

WANT: $\{c_i\}$ s.t. $\bigoplus_{i=1}^{n} c_i = c = a \oplus b$

$c_i = a_i \oplus b_i$

AND gates:



GIVEN: $\bigoplus_{i=1}^{n} a_i = a$ $\quad \bigoplus_{i=1}^{n} b_i = b$

WANT: $\{c_i\}$ s.t. $\bigoplus_{i=1}^{n} c_i = c = a \cdot b$

$c_i = ?$

$$a \cdot b = \left(\sum_{i=1}^{n} a_i\right) \cdot \left(\sum_{i=1}^{n} b_i\right) \mod 2$$

$$= \left(\sum_{i=1}^{n} a_i \cdot b_i\right) + \left(\sum_{i \ne j} a_i \cdot b_j\right) \mod 2$$

↑ P_i locally $\qquad$ ↑ Reshare

Outputs:

For each output wire w:

Each party P_i holds a random share $v_i^w \in \{0,1\}$

↳ Sends v_i^w to all parties

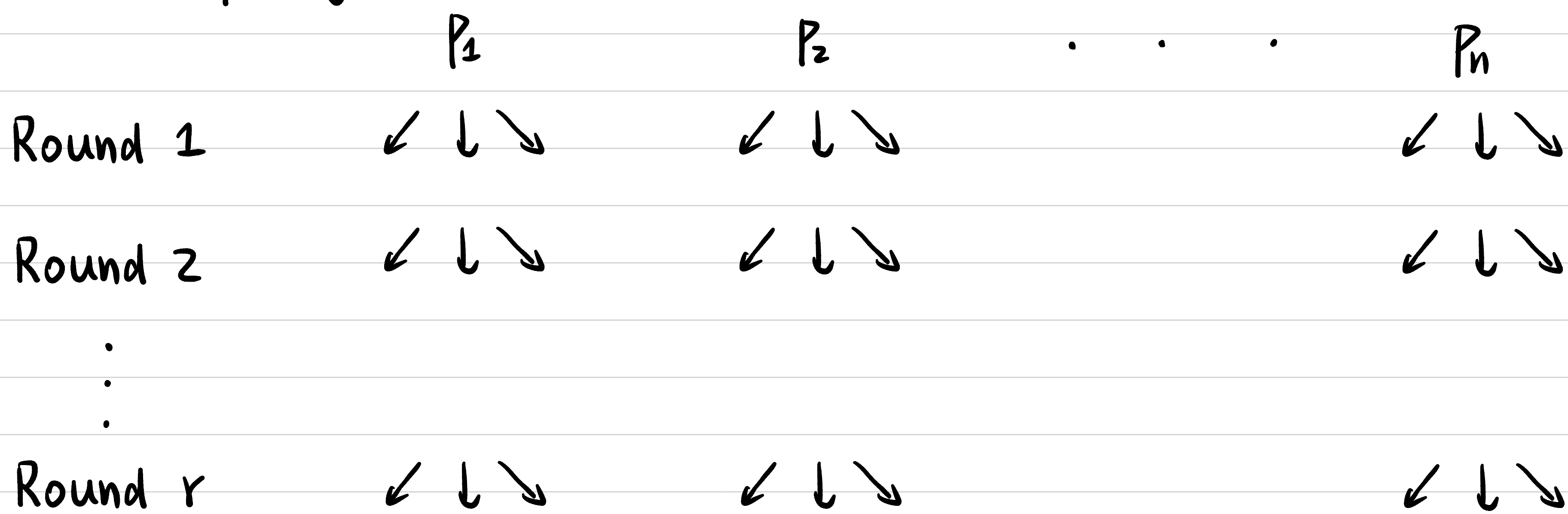
Each party computes the value $v^w = \bigoplus_{i=1}^{n} v_i^w$

MPC for any function with $t \leq n-1$ (GMW)

Computational Complexity?

Communication Complexity?

Round Complexity?



GMW Compiler

Given a semi-honest protocol:

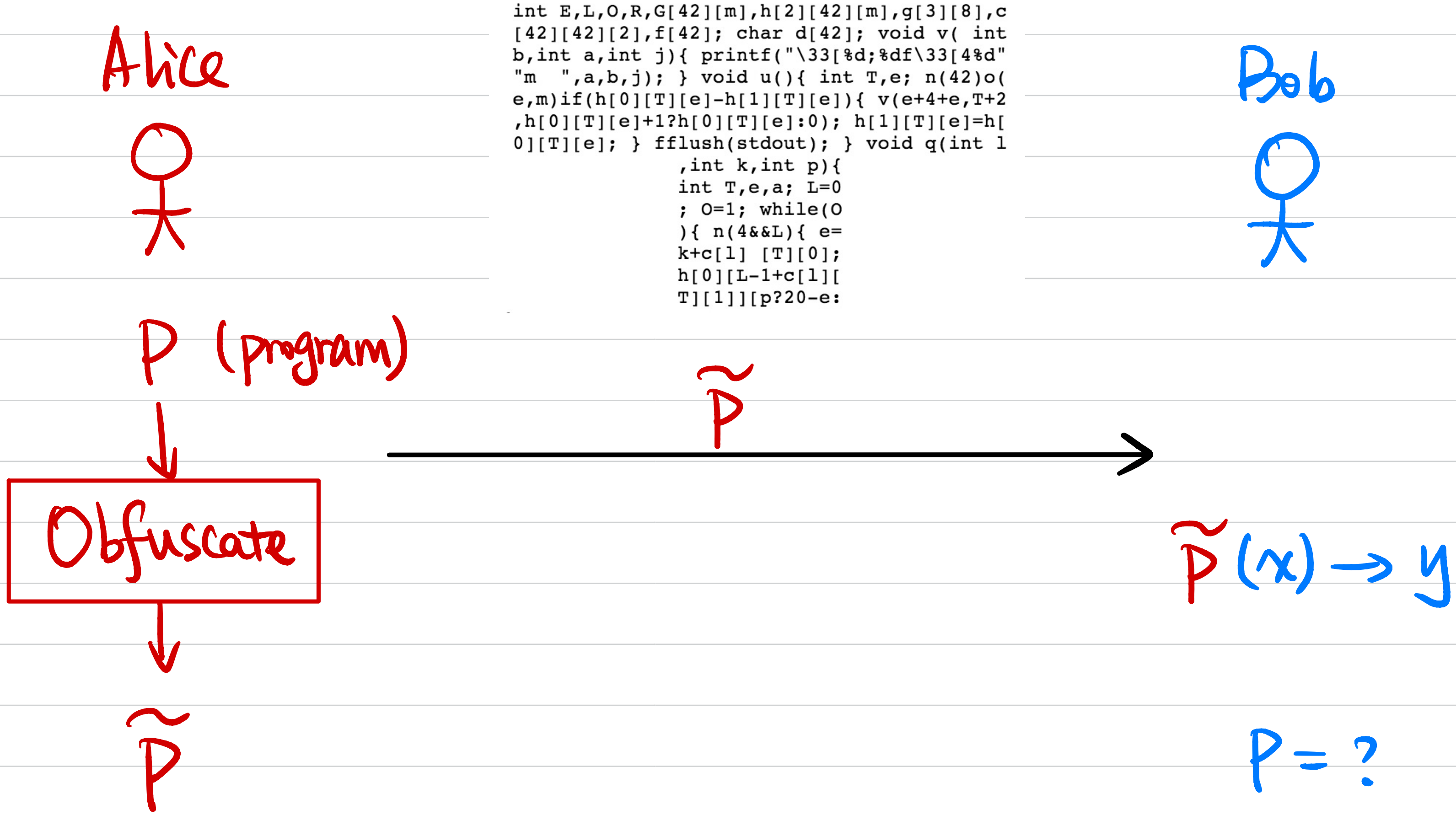
Once inputs & randomness are fixed, protocol is deterministic.

Step 1: Each party P_i commits to its input x_i & randomness r_i to be used in the semi-honest protocol.

Step 2: Run semi-honest protocol.

Along with every message, prove in ZK that the message is computed correctly (based on its input, randomness, transcript so far)

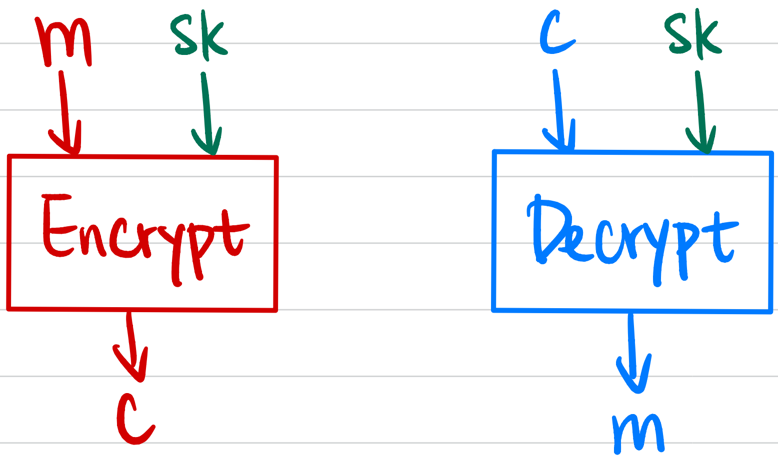
Program Obfuscation




```
int E,L,O,R,G[42][m],h[2][42][m],g[3][8],c
[42][42][2],f[42]; char d[42]; void v( int
b,int a,int j){ printf("\33[%d;%df\33[4%d"
"m  ",a,b,j); } void u(){ int T,e; n(42)o(
e,m)if(h[0][T][e]-h[1][T][e]){ v(e+4+e,T+2
,h[0][T][e]+1?h[0][T][e]:0); h[1][T][e]=h[
0][T][e]; } fflush(stdout); } void q(int l
              ,int k,int p){
              int T,e,a; L=0
              ; O=1; while(O
              ){ n(4&&L){ e=
              k+c[l] [T][0];
              h[0][L-1+c[l][
              T][1]][p?20-e:
```

Goal: Make the program "unintelligible" without affecting its functionality.

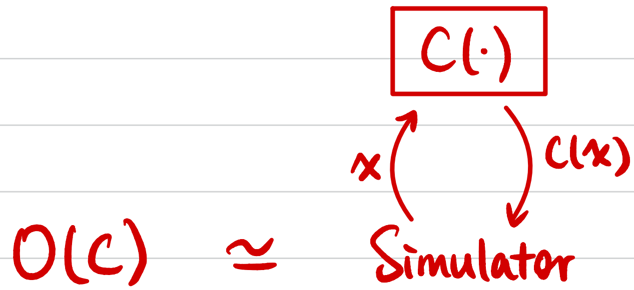
Symmetric-Key to Public-Key



Formal Definition: Virtual Black Box (VBB)

Obfuscator O: $C \xrightarrow{O} O(C)$

- **Functionality:** $O(C)$ computes the same function as C.
- **Polynomial Slowdown:** $|O(C)| \leq \text{poly}(n) \cdot |C|$
- **Security (Virtual Black Box):**
  $\forall$ PPT A, $\exists$ PPT S, s.t. $\forall C$, $A(O(C)) \overset{c}{\simeq} S^{C(\cdot)}(1^{|C|})$.



Thm VBB obfuscator for all poly-sized circuits is impossible to achieve.

$$C(x) := \begin{cases} b & \text{if } x = a \\ m & \text{if } x(a) = b \\ 0 & \text{otherwise} \end{cases}$$

Formal Definition: Indistinguishability Obfuscation (iO)

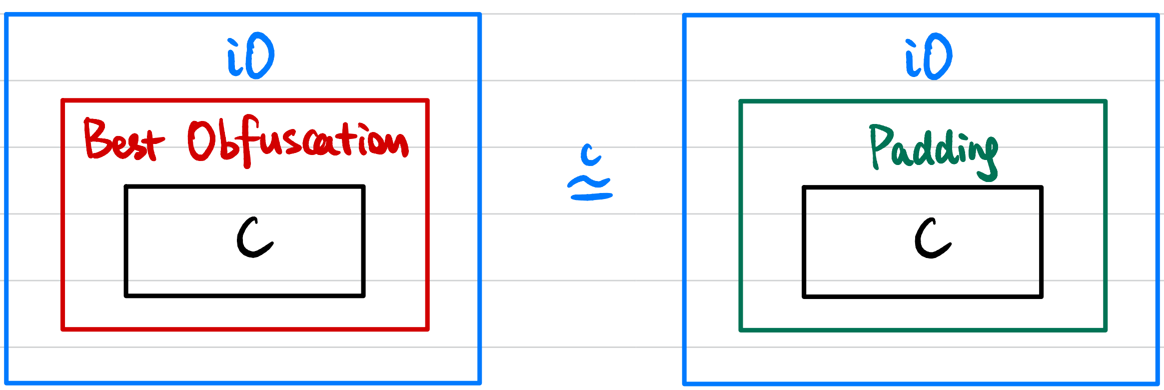
Obfuscator O: $C \xrightarrow{O} O(C)$

- **Functionality:** $O(C)$ computes the same function as C.
- **Polynomial Slowdown:** $|O(C)| \leq poly(n) \cdot |C|$
- **Security (indistinguishability obfuscation):**

  If C_0 & C_1 compute the same function and $|C_0| = |C_1|$,

  then $O(C_0) \overset{c}{\approx} O(C_1)$

- **Best Possible Obfuscation**



PKE from iO

Let $G: \{0,1\}^n \to \{0,1\}^{2n}$ be a length-doubling PRG.

- $Gen(1^n)$:

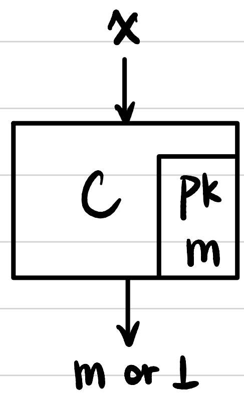
  $sk \xleftarrow{\$} \{0,1\}^n$

  $pk := G(sk)$

- $Enc_{pk}(m)$:

  $$C_{pk,m}(x) := \begin{cases} m & \text{if } G(x) = pk \\ \perp & \text{otherwise} \end{cases}$$

  Output $c \leftarrow iO(C_{pk,m})$



- $Dec_{sk}(c)$: ?