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)

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

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)

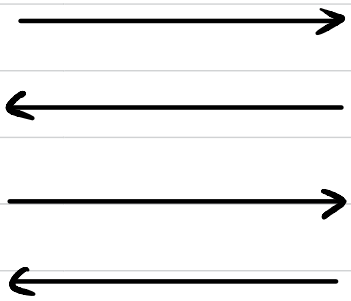
necessary

Oblivious Transfer (OT)

Sender

Receiver

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



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

Output: $\perp$

Output: m_c

Oblivious Transfer (OT)

Cyclic group G of order q with generator g

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

Sender		Receiver
Input: $m_0, m_1 \in \{0,1\}^l$		Input: $c \in \{0,1\}$
$a \xleftarrow{\$} \mathbb{Z}_q$	$\xrightarrow{A = g^a}$	$b \xleftarrow{\$} \mathbb{Z}_q$
	$\xleftarrow{B = g^b \cdot A^c}$	
$k_0 := H(B^a)$ $k_1 := H((\frac{B}{A})^a)$		
	$\xrightarrow{\begin{array}{l} ct_0 := k_0 \oplus m_0 \\ ct_1 := k_1 \oplus m_1 \end{array}}$	
		Output: $m_c := ct_c \oplus H(A^b)$

Thm If CDH is hard in 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: $m_0, m_1 \in \{0,1\}^\ell$

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

$A = g^a$ →

← $B \xleftarrow{\$} G$

$k_0 := H(B^a)$

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

$ct_0 := k_0 \oplus m_0$

$ct_1 := k_1 \oplus m_1$ →

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

Receiver

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

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

$A = g^a$ →

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

← $B = g^b \cdot A^c$

$k_c := H(g^{ab})$

$ct_c := k_c \oplus m_c$

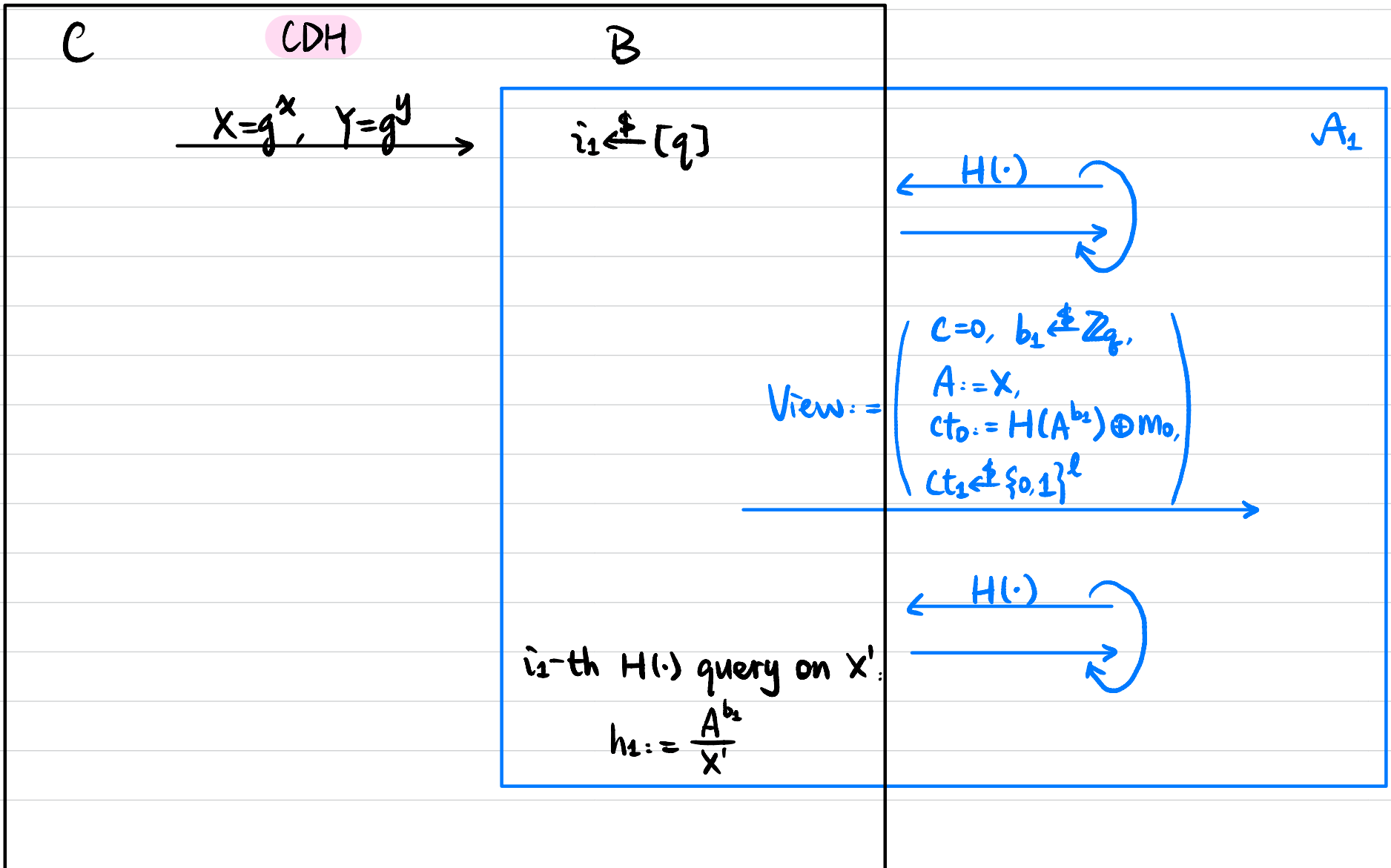
$ct_{1-c} \xleftarrow{\$} \{0,1\}^\ell$ →

Output: ?

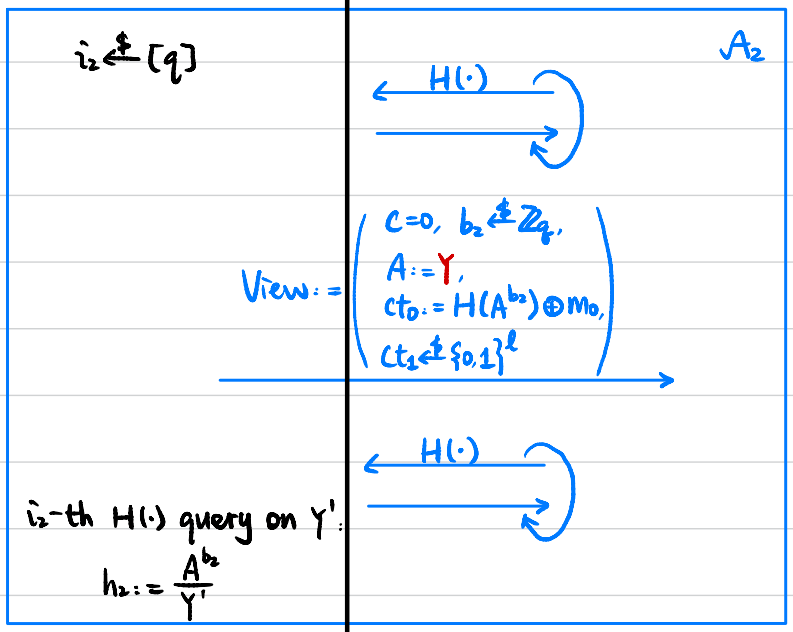
$S_B(1^n, c, m_c) \simeq \text{View}_R^{\Pi}((m_0, m_1), c, n)$

Assume $\exists$ PPT A that can distinguish. A must be querying $H(g^{ab-a^2})$ when $c=0$ or $H(g^{ab+a^2})$ when $c=1$ with non-negligible probability. WLOG assume $c=0$.

We construct PPT B to break CDH in the random oracle model.

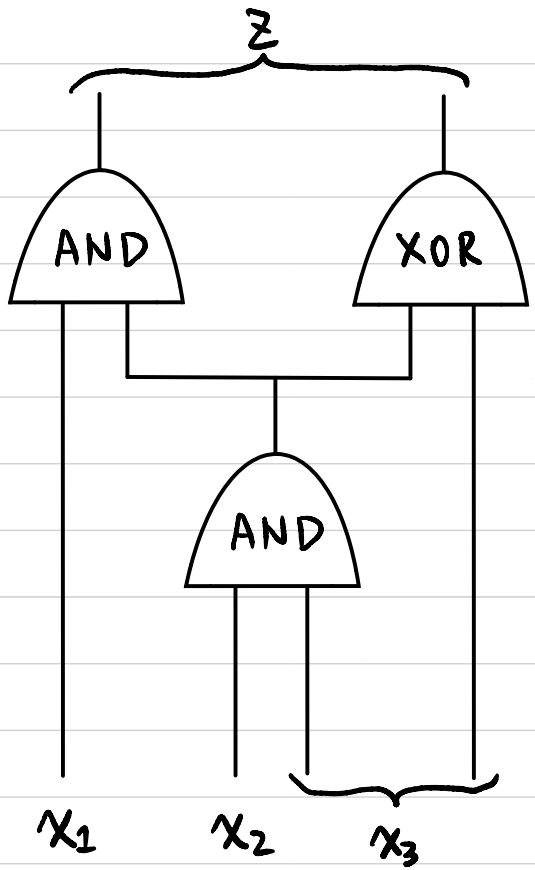


B (continued)

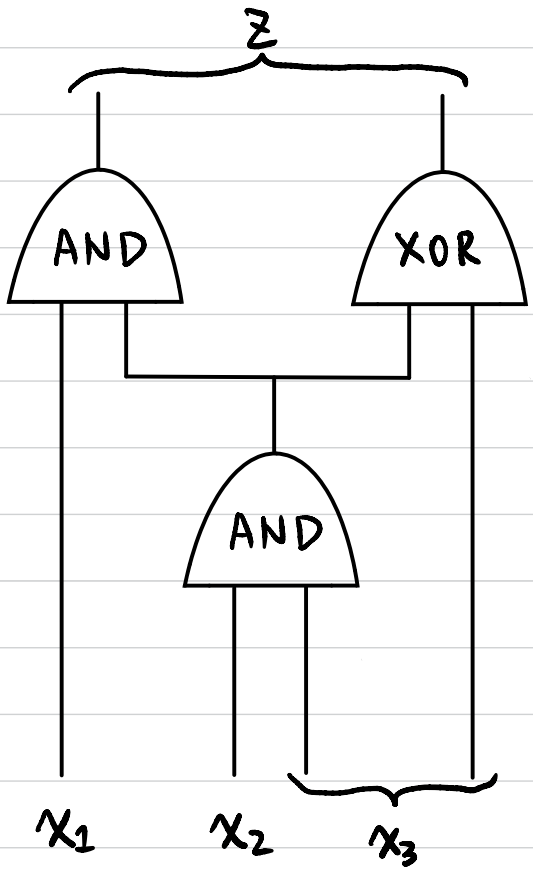


Output $\left(\frac{h_3}{h_1 \cdot h_2}\right)^{2^{-1}}$

Arbitrary Function → Represent it as a Boolean circuit



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



Throughout the protocol, we keep the invariant:

For each wire w:

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

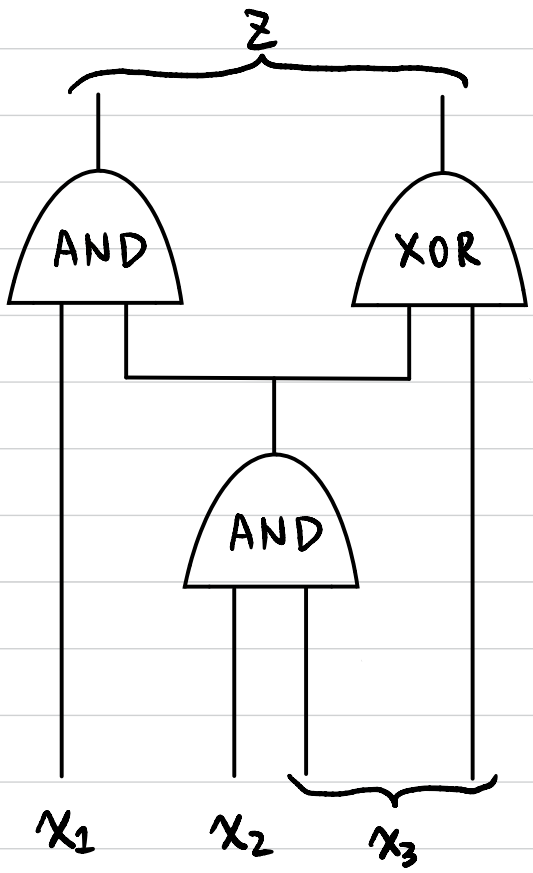
then the n parties hold an additive secret share of v^w

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 $t \leq n-1$ (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:

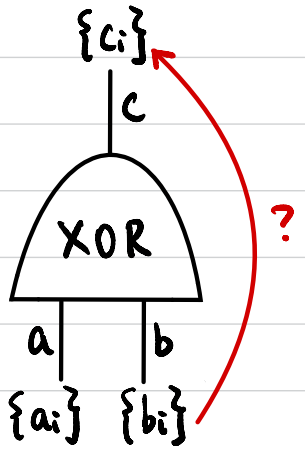
For each input wire w:

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:

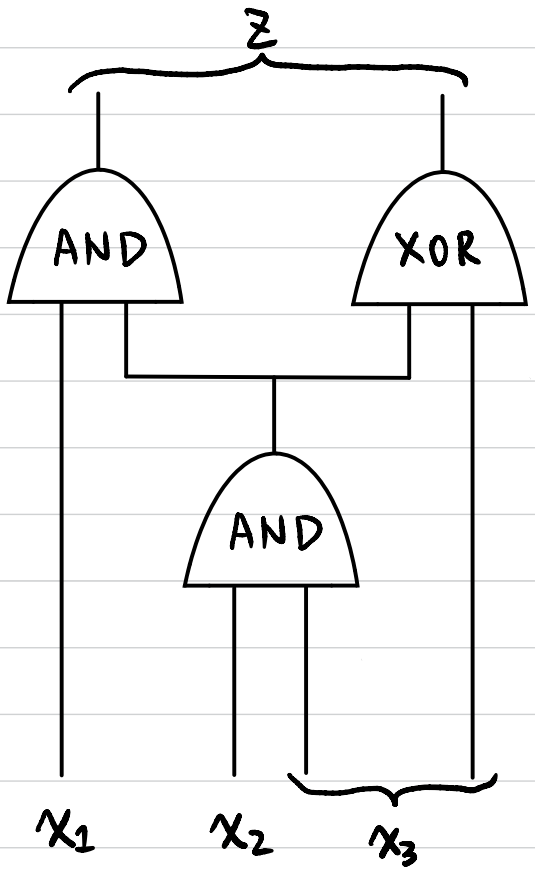


GIVEN: $\bigoplus_{i=1}^{n} a_i = a$ $\bigoplus_{i=1}^{n} b_i = b$

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

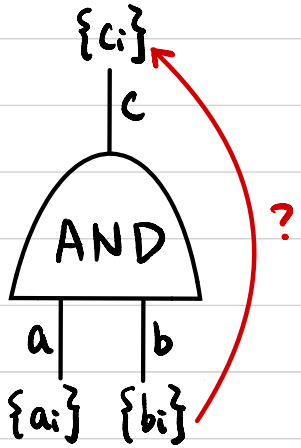
$c_i = a_i \oplus b_i$

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

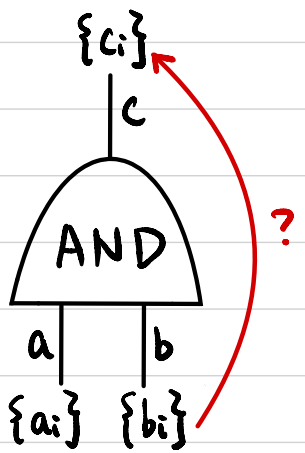
Each party P_i holds a random share $v_i^w \in \{0,1\}$

→ 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 $t \leq n-1$ (GMW)

AND gates:



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

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

$c_i = ?$

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

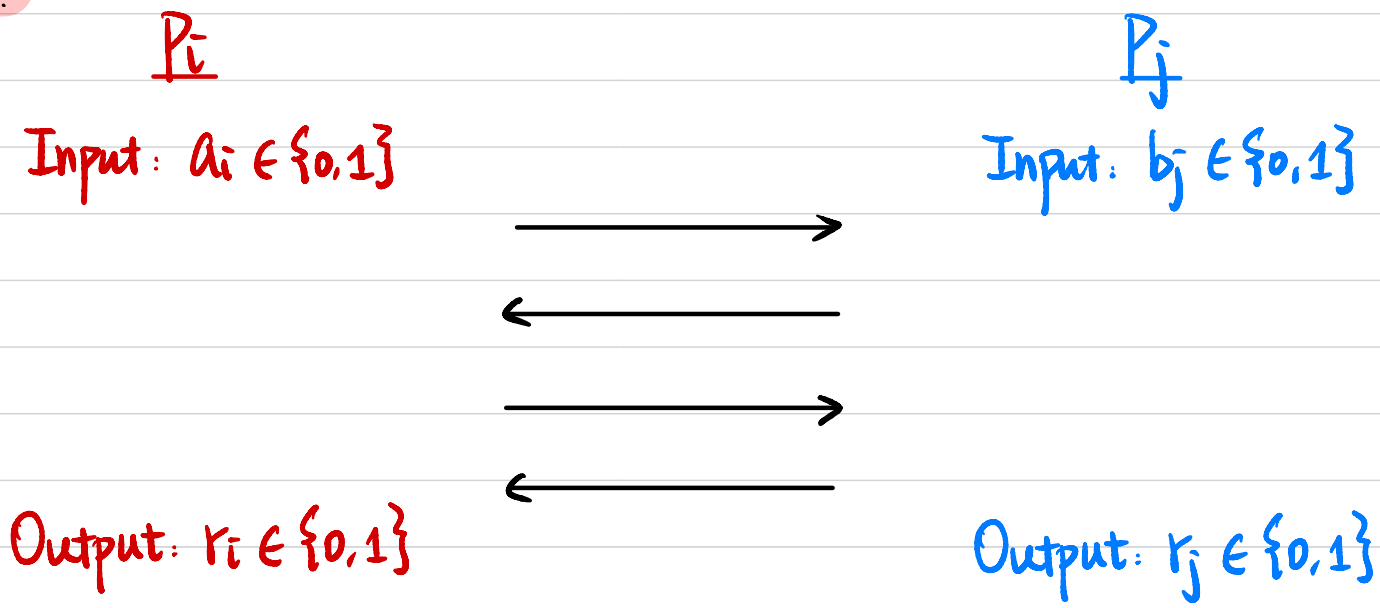
$$= \left(\sum_{i=1}^{n} a_i \cdot b_i\right) + \left(\sum_{i \neq j} a_i \cdot b_j\right) \mod 2$$

$a_i \cdot b_i$: P_i locally

$a_i \cdot b_j$: ?

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

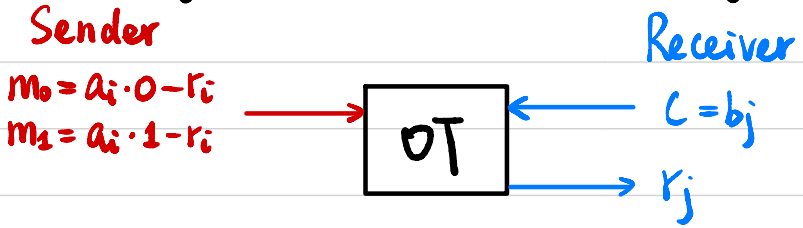
Reshare:



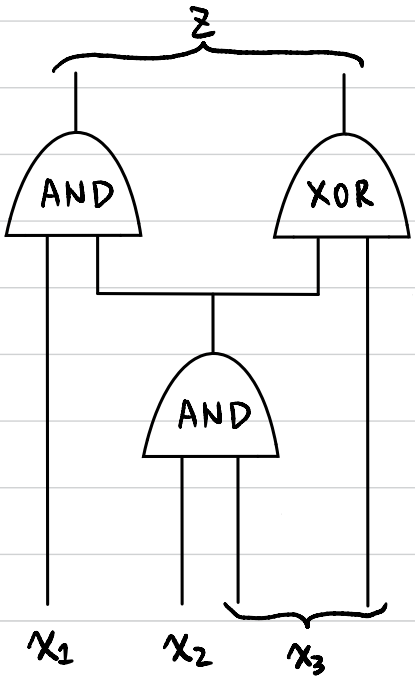
WANT: Random $r_i, r_j \in \{0,1\}$ s.t. $r_i + r_j = a_i \cdot b_j \pmod 2$

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 $t \leq n-1$ (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:

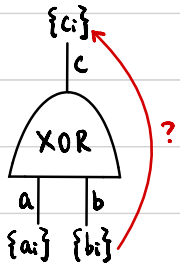
For each input wire w:

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:

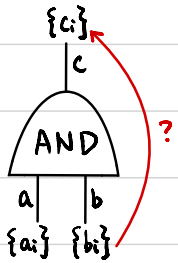


GIVEN: $\bigoplus_{i=1}^{n} a_i = a$ $\bigoplus_{i=1}^{n} b_i = b$

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

$c_i = a_i \oplus b_i$

AND gates:



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

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

$c_i = ?$

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

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

P_i locally — 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?

Each party: $O(\#\text{AND} \cdot n + \#\text{XOR})$

Communication Complexity?

Each party: $O(\#\text{AND} \cdot n)$

Round Complexity?

$O(\text{depth of AND gates})$

GMW Compiler

Given a semi-honest protocol:

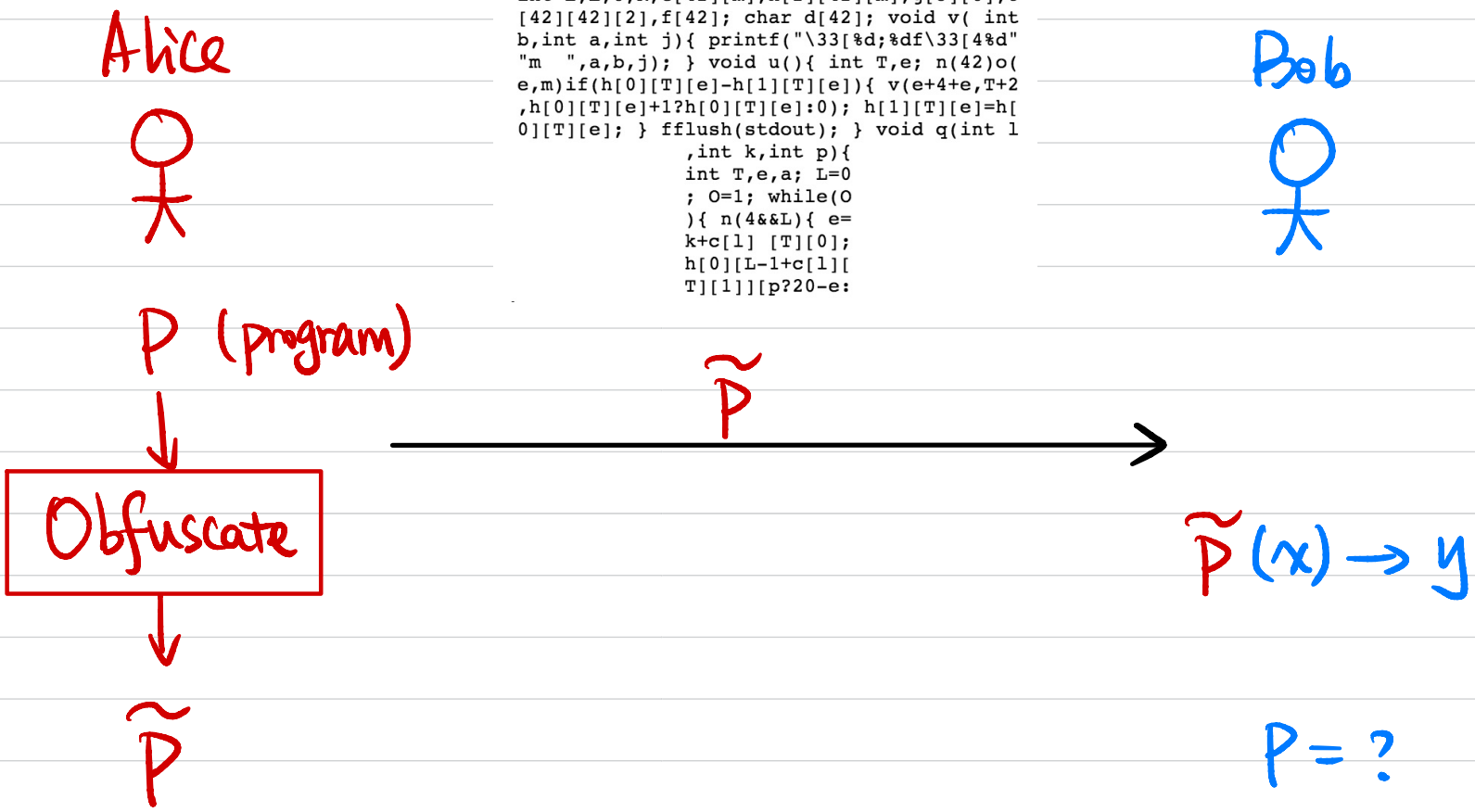
Once inputs & randomness are fixed, protocol is deterministic.

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.