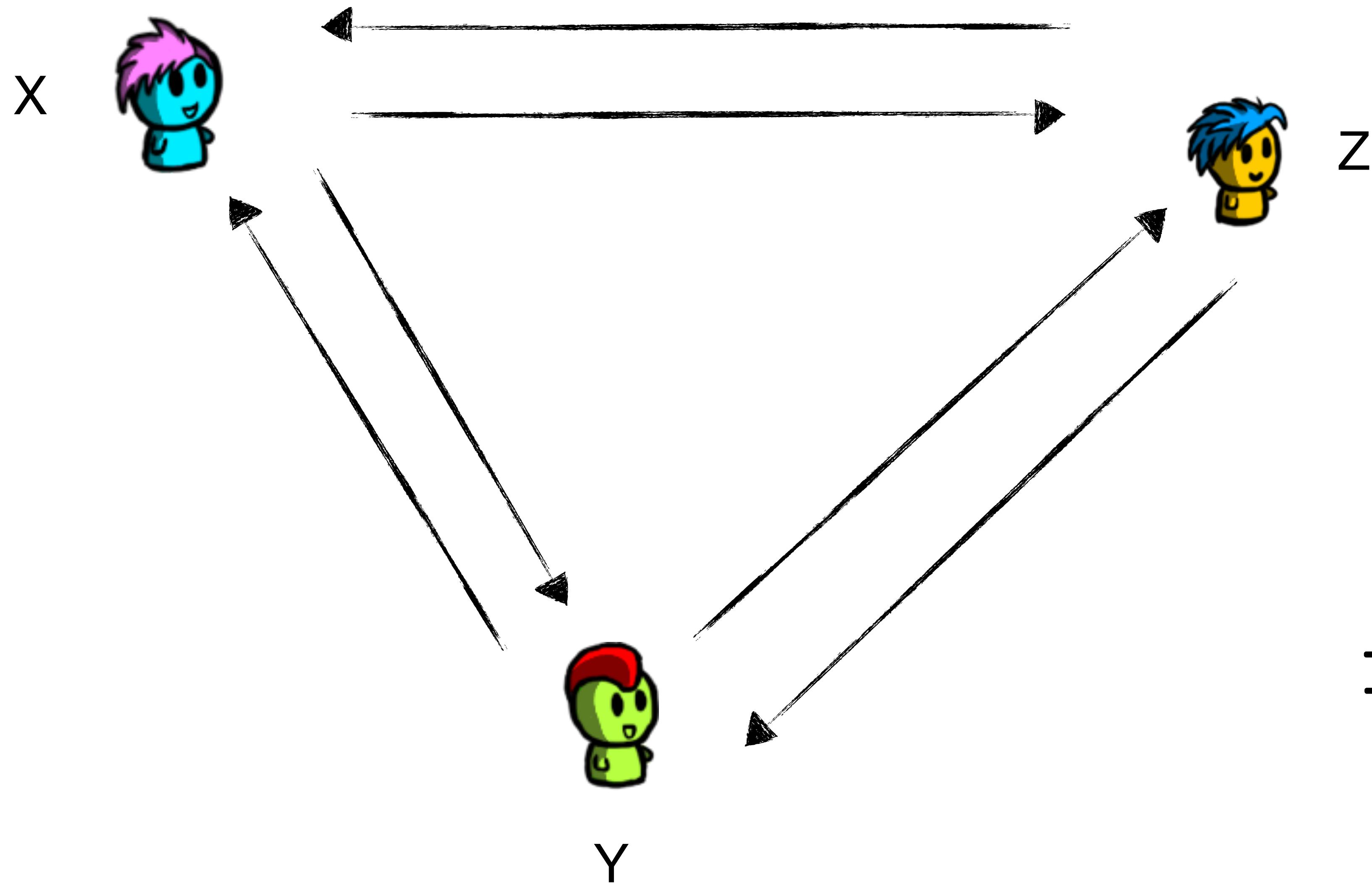




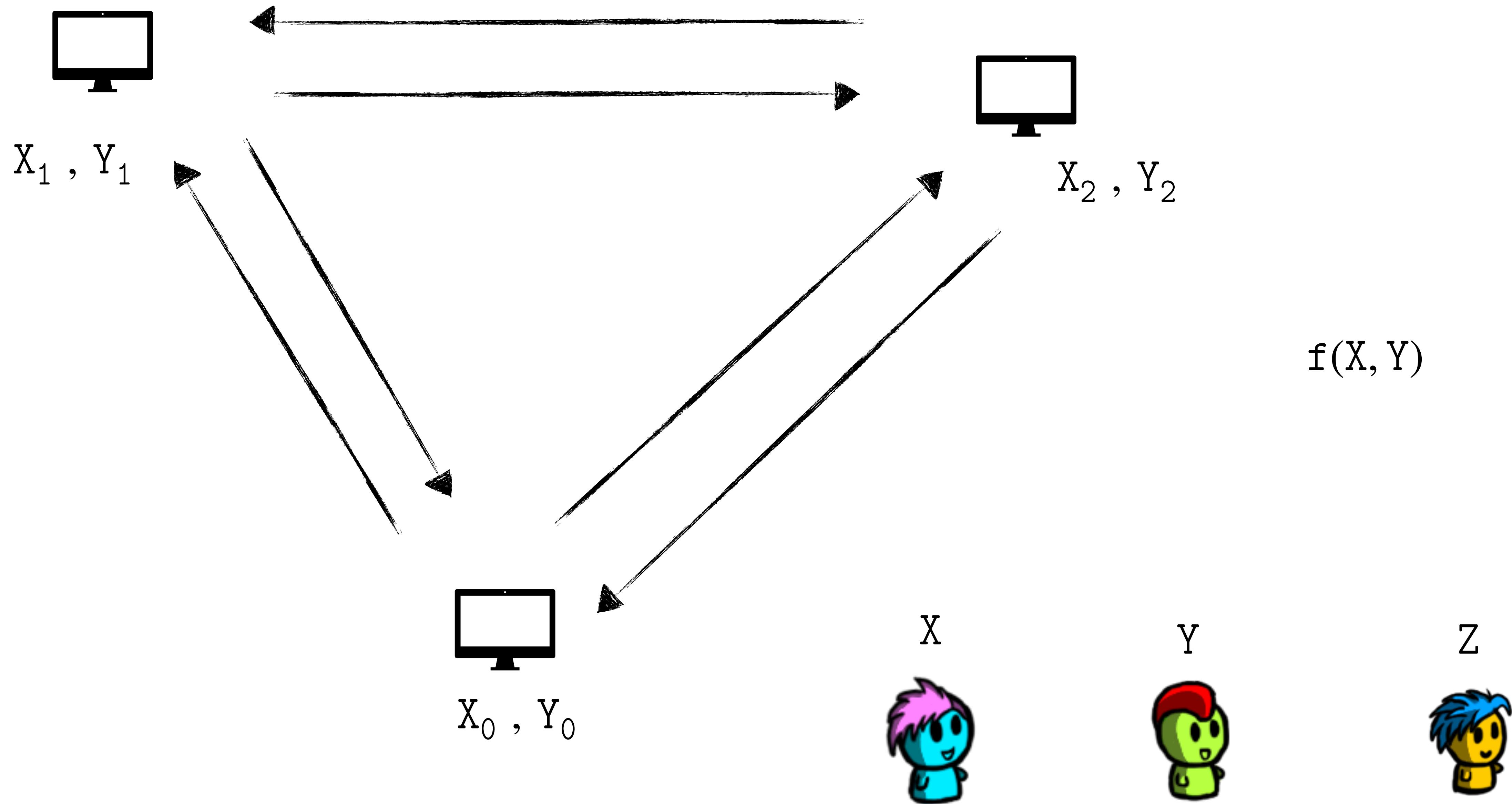
# CS670: Cryptographic Techniques for Privacy Preservation

## Module 1: Secure Multiparty Computation

**Adithya Vadapalli**



$$f(X, Y, Z)$$



# Properties of MPC

Expressibility

Minimum # Parties

Threat Model

Maximum # Adversarial Parties

Performance

# Expressibility

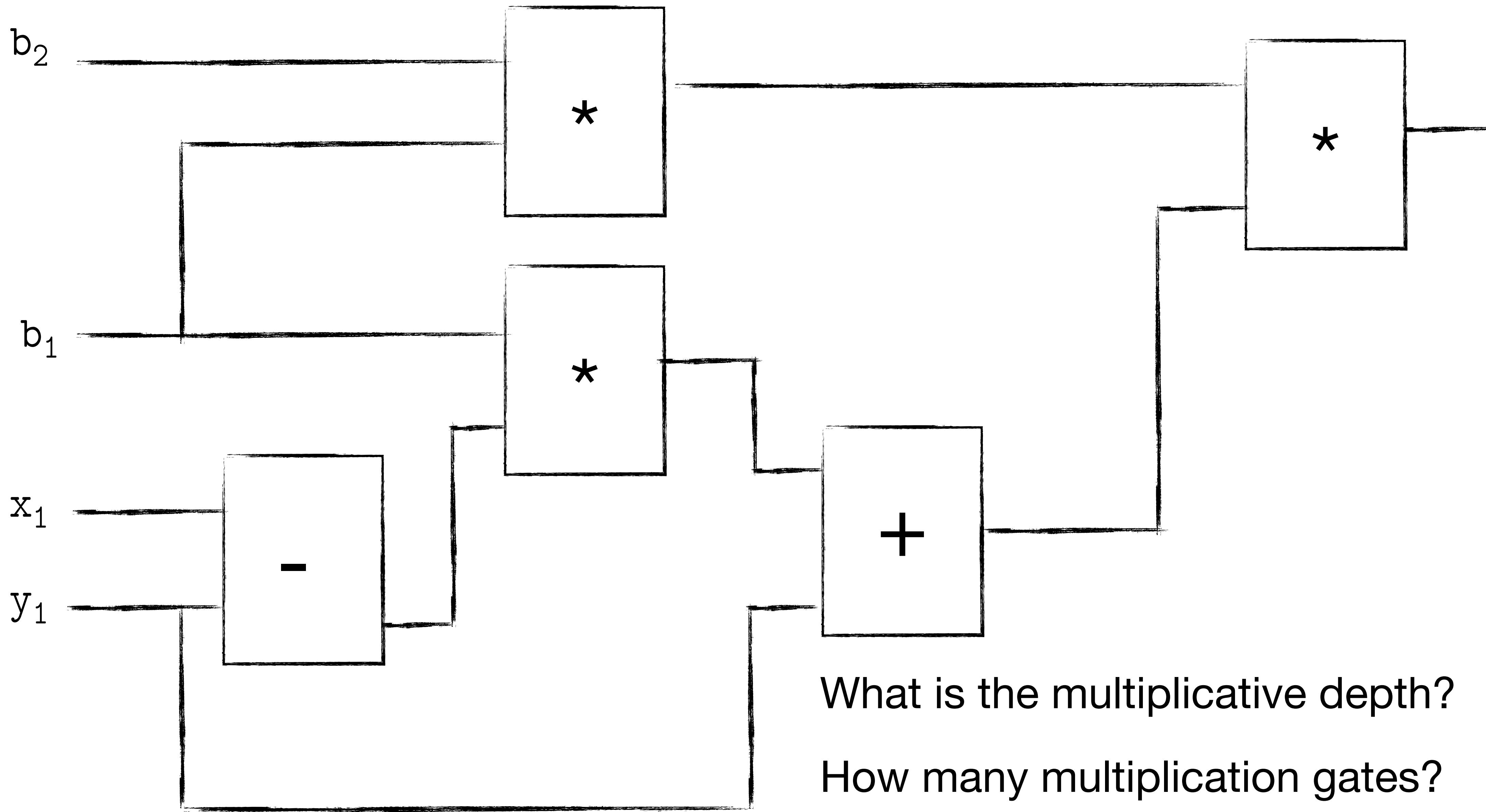
What kind of functions can the MPC protocol evaluate?

Generic

Some MPC protocols can compute any function

Specific

Some MPC protocols are only for specific function



# Types of Gates

Linear Gates  Typically very easy to compute



$x_0$   $y_0$



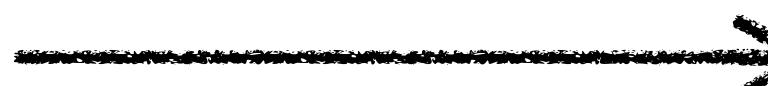
$x_1$   $y_1$

hold shares of  $x$  and  $y$

How can they get shares of:  $\alpha \cdot x + \beta \cdot y$

$$z_0 \leftarrow \alpha \cdot x_0 + \beta \cdot y_0$$

$$z_1 \leftarrow \alpha \cdot x_1 + \beta \cdot y_1$$

Non-Linear Gates  More complicated

# Min # Parties

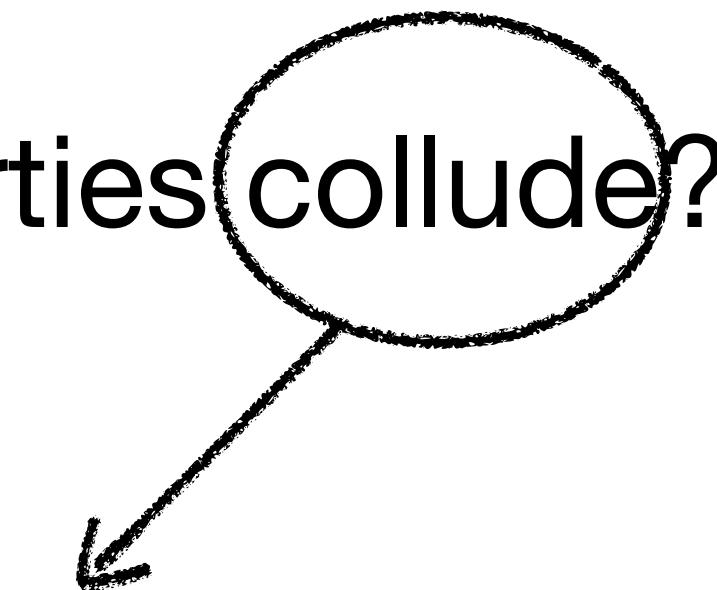
How many computational parties are needed to run the protocol?

The larger the value, the more challenging it is to deploy the protocol.

A protocol that requires, for example, 3 Parties is called a 3PC

# Threat Model

Does the protocol remain secure if some of the parties **collude**?



*Two parties collude if they share with each other the secret values they are not supposed to.*

# Max # Adversarial Parties

How many parties in the protocol can be adversarial?

Honest Majority

Honest Minority

# Performance

Local Computation • Round Complexity • Bandwidth Consumption

Which is the most important parameter?

Depends on the deployment scenario.

# Secure Multiparty Computation

(On Secret Shares)



$x_0, y_0$



$x_1, y_1$

$$x_0 + x_1 = x$$

$$y_0 + y_1 = y$$

Alice and Bob hold additive shares of:  $X$  and  $y$

Their goal is to get additive shares of:  $x + y$

This is basically free (*in terms of communication*)

$$z_0 \leftarrow x_0 + y_0$$



$$z_1 \leftarrow x_1 + y_1$$



# Secure Multiparty Computation

(On Additive Secret Shares)



$x_0, y_0$



$x_1, y_1$

$$x_0 + x_1 = x$$

$$y_0 + y_1 = y$$

Alice and Bob hold additive shares of:  $X$  and  $y$

Their goal is to get additive shares of:  $x \cdot y$

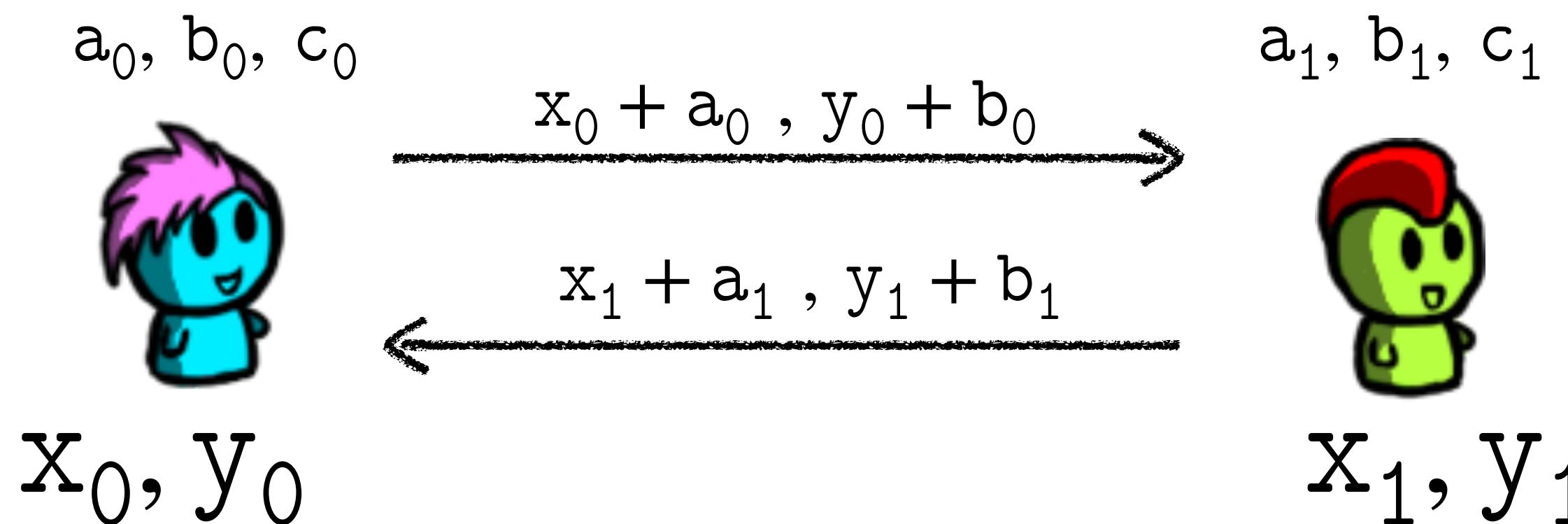
# Secure Multiparty Computation

(On Additive Secret Shares)



Beaver's Triples:

Distribute shares of random inputs ( $a$  and  $b$ ) and their product  $c$



$$x_0 + x_1 = x$$

$$y_0 + y_1 = y$$

Goal:  $z_0$

$z_1$  such that

$z_0 + z_1 = z$

$$\alpha \leftarrow x + a$$

$$\beta \leftarrow y + b$$

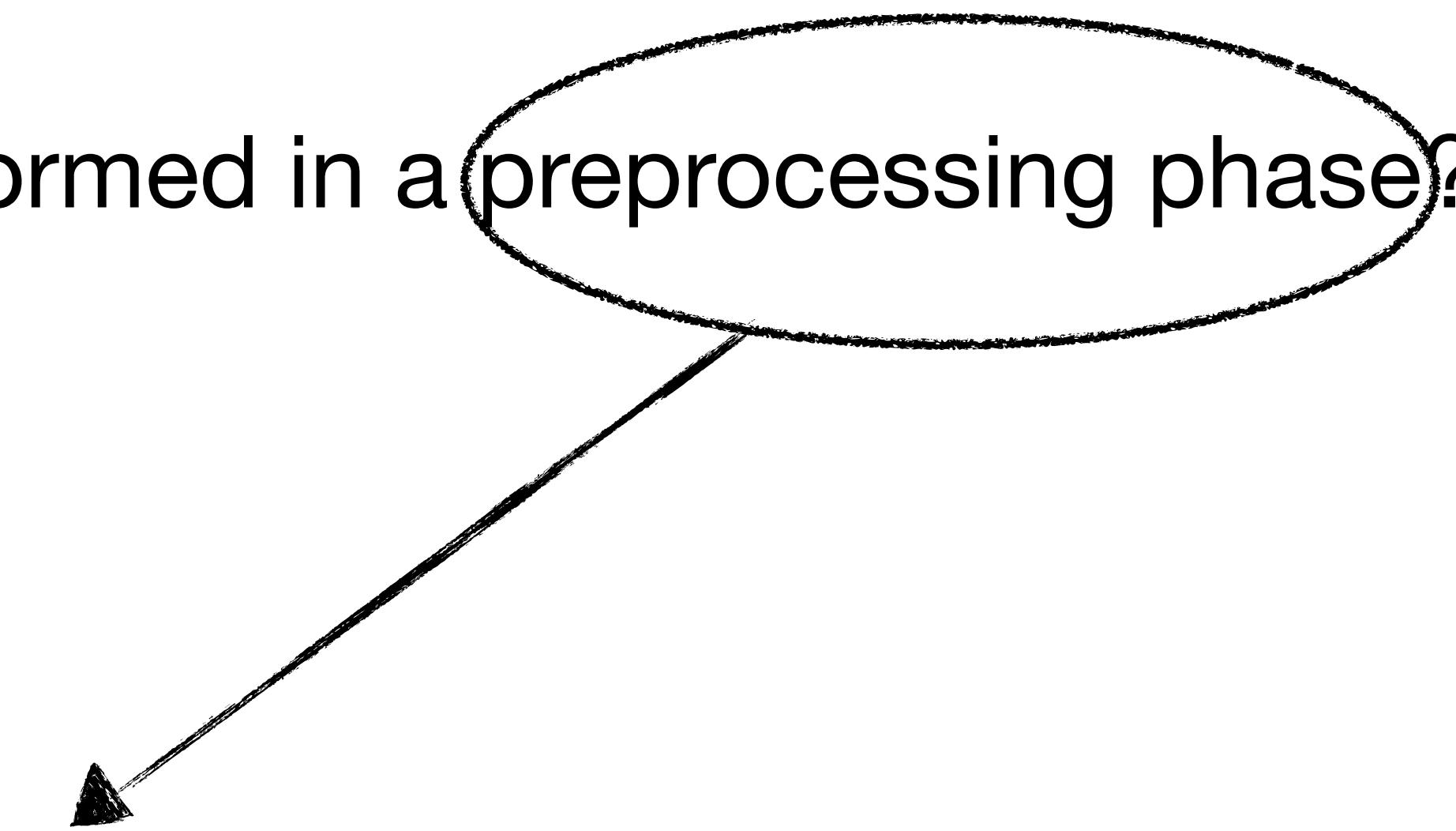
public values

$$z_0 \leftarrow \alpha \cdot y_0 - \beta \cdot a_0 + c_0$$

$$z_1 \leftarrow \alpha \cdot y_1 - \beta \cdot a_1 + c_1$$

# Secure Multiparty Computation

Can any of the operations be performed in a preprocessing phase?



Can be done ahead of time before we know the actual inputs.

# Secure Multiparty Computation

## Preprocessing Phase      vs

## Online Phase

It can happen before the actual inputs are known

Happens when the actual inputs are known

It is okay if this process is somewhat expensive

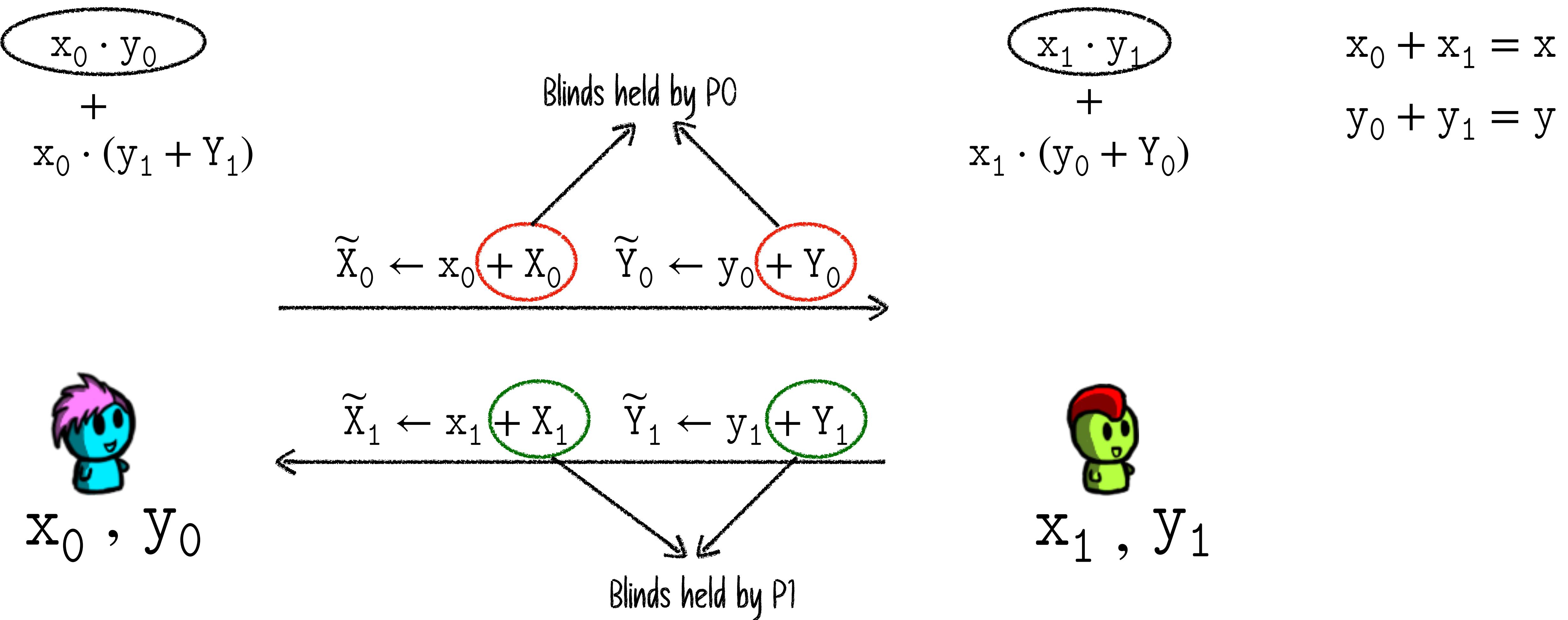
Ideally you'd like it to be as efficient as possible

In Multiplication: Beaver Triples

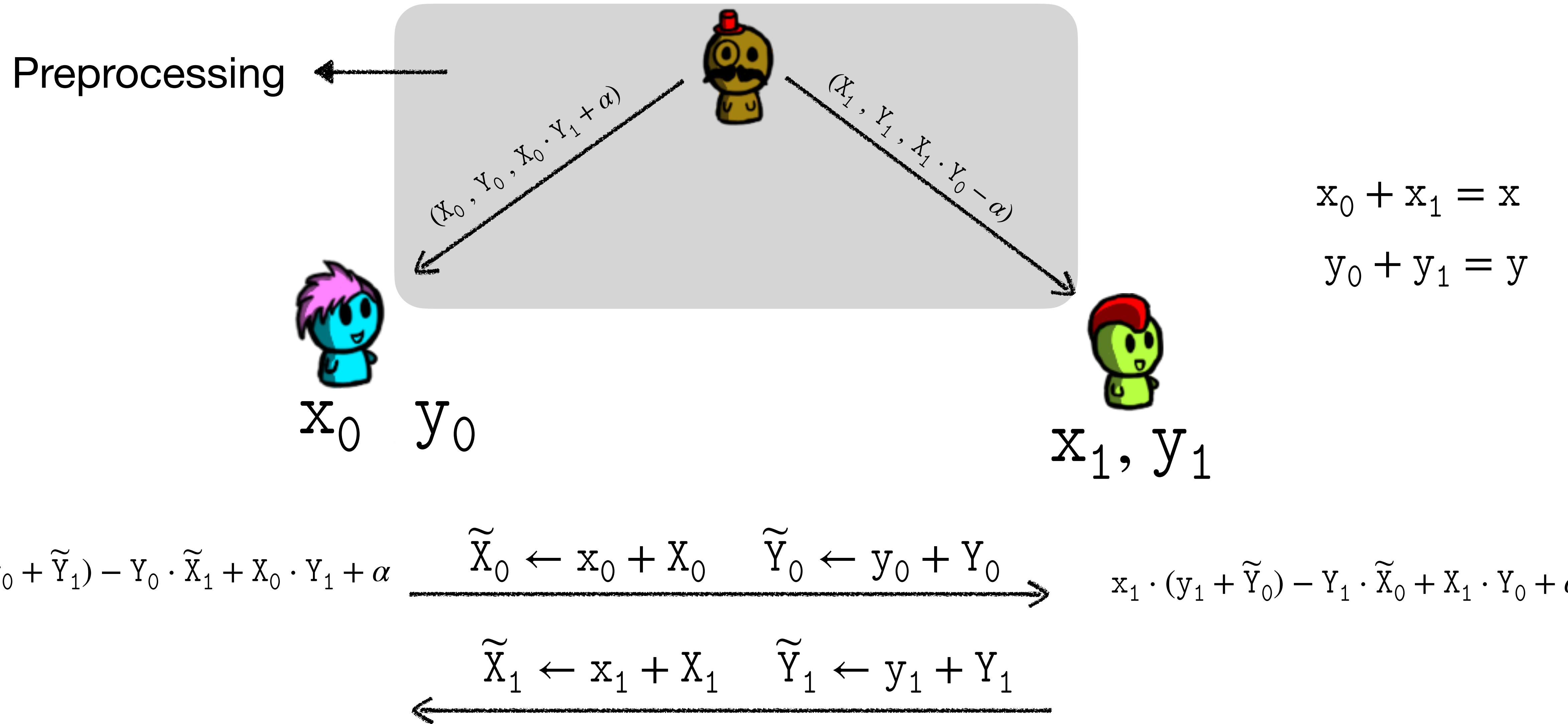
The exchange of blinds and computation of the answer

# Secure Multiparty Computation

(On Additive Secret Shares)



# Secure Multiparty Computation $\wedge$ (On Additive Secret Shares)



# Properties of the Protocol

Expressibility: generic

Minimum number of parties: 2

Threat Model: Semi-honest

Maximum number of adversarial parties: 1

Performance (g total gates, m multiplication gates, multiplication depth d):

Local Computation:  $\mathcal{O}(g)$

Total Communication: 6m (preprocessing) + 2m per party

Round Complexity: 1 preprocessing + d

# Secure Multiparty Computation

(On Replicated Secret Shares)



$x_0, x_1$

$y_0, y_1$



$x_1, x_2$

$y_1, y_2$



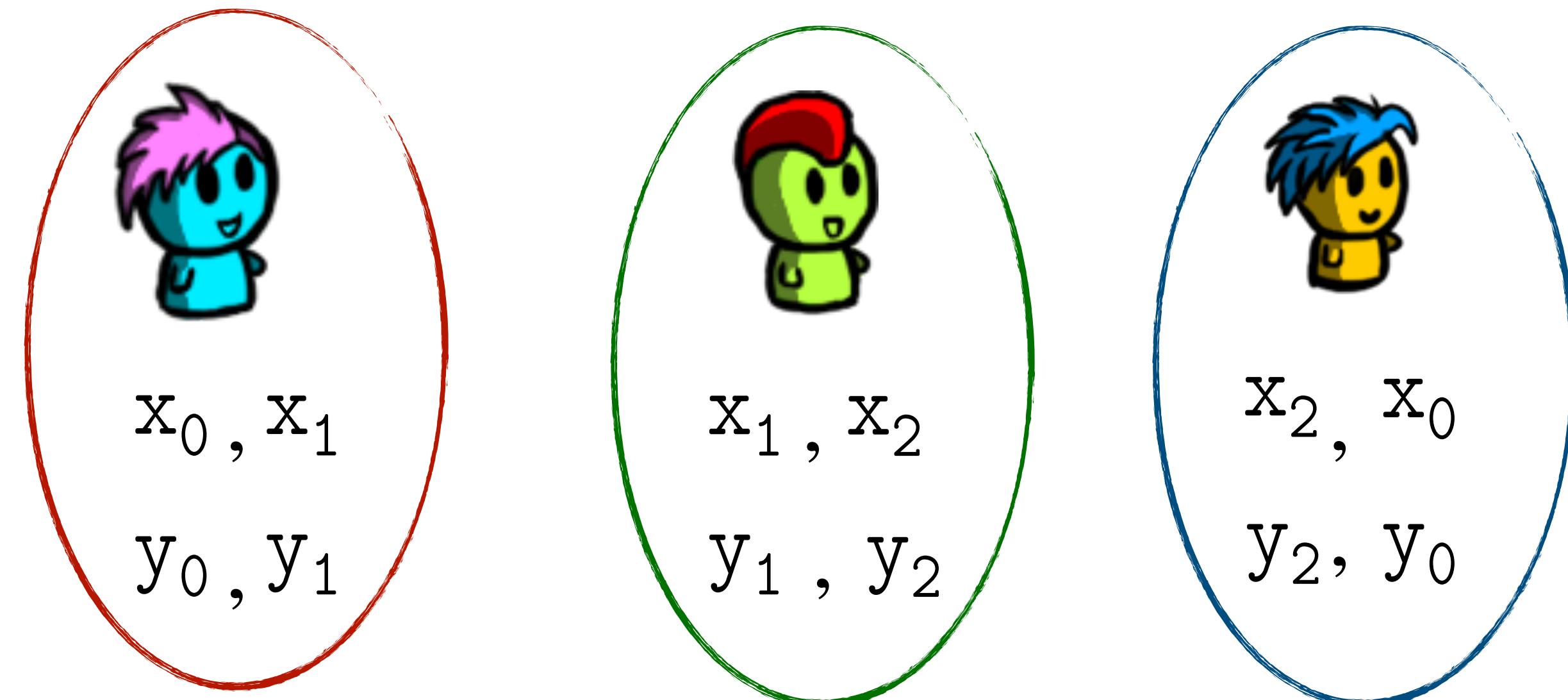
$x_3, x_0$

$y_3, y_0$

$$z_0 + z_1 + z_2 = (x_0 + x_1 + x_2) \cdot (y_0 + y_1 + y_2)$$

# Secure Multiparty Computation

(On Replicated Secret Shares)



$$z_0 + z_1 + z_2 = (x_0 + x_1 + x_2) \cdot (y_0 + y_1 + y_2)$$

What is the problem with this?

$$\textcolor{blue}{x_0 \cdot y_0} + \textcolor{red}{x_0 \cdot y_1} + \textcolor{blue}{x_0 \cdot y_2}$$

+

$$\textcolor{red}{x_1 \cdot y_0} + \textcolor{red}{x_1 \cdot y_1} + \textcolor{green}{x_1 \cdot y_2}$$

+

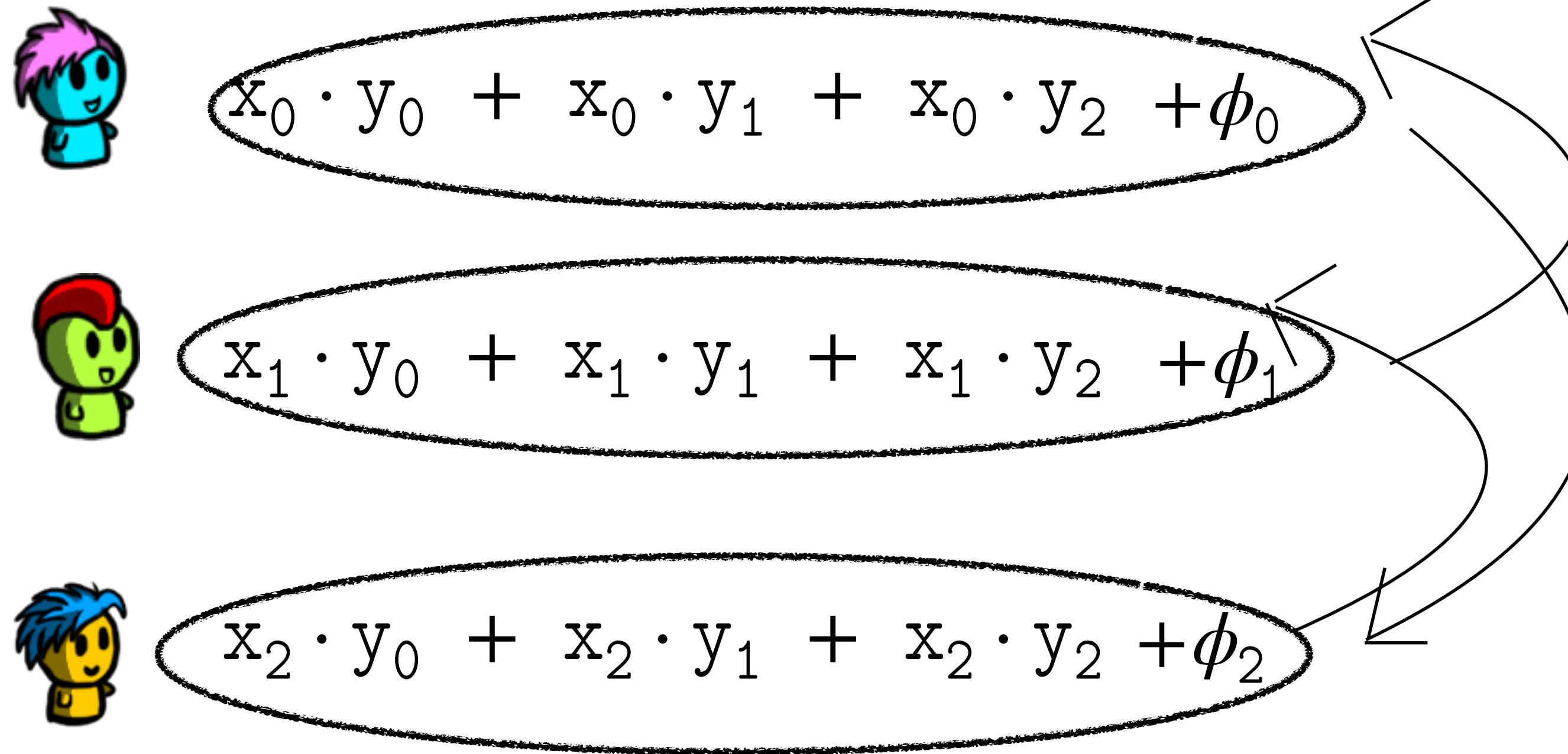
$$\textcolor{blue}{x_2 \cdot y_0} + \textcolor{green}{x_2 \cdot y_1} + \textcolor{green}{x_2 \cdot y_2}$$

# **Acadly Attendance Check**

# Zero Sharing

(On Replicated Secret Shares)

Parties can compute shares of zero:  $\phi_0 + \phi_1 + \phi_2 = 0$



# Zero Sharing

(On Replicated Secret Shares)

How do you get zero shares?

PRGs

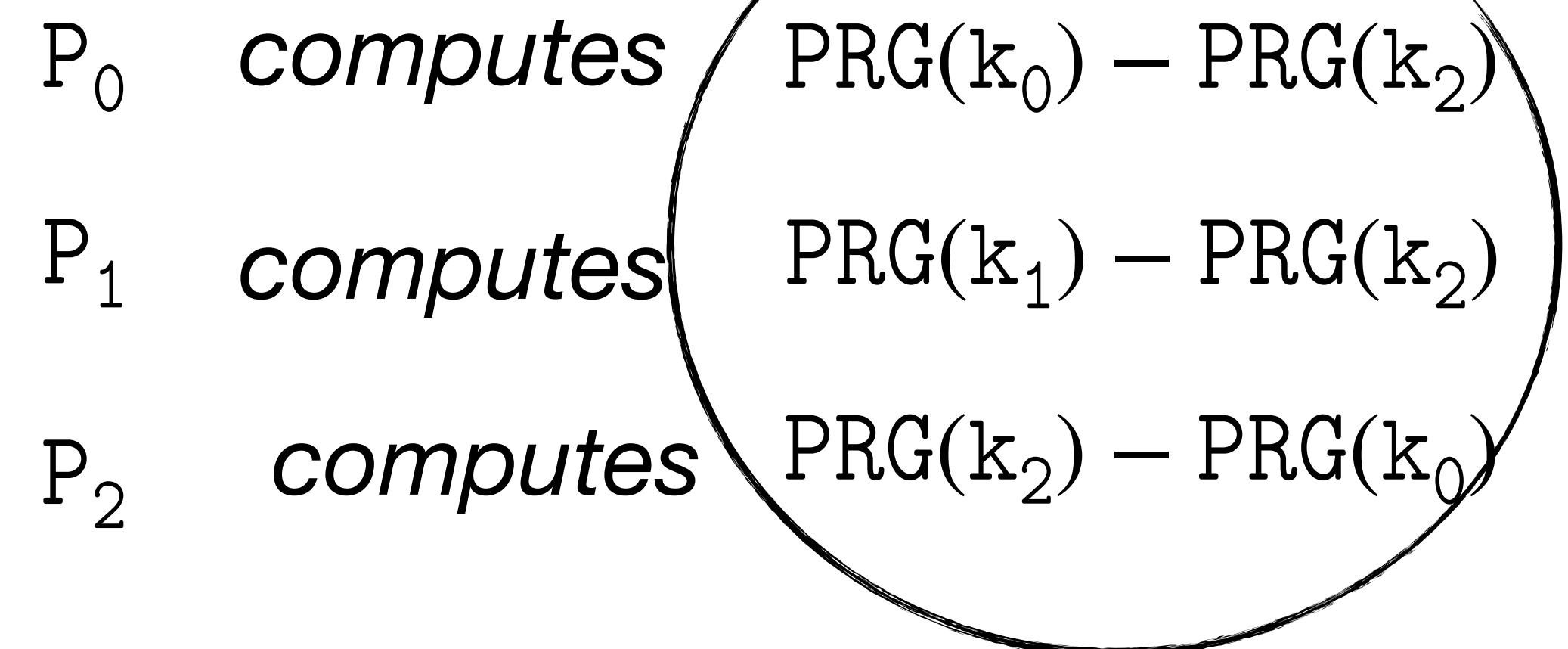
Each party  $i$  picks a random key  $k_i$

In the preprocessing phase:

$$P_0 \xrightarrow{k_0} P_1$$

$$P_1 \xrightarrow{k_1} P_2$$

$$P_2 \xrightarrow{k_2} P_0$$



# Properties of the Protocol

Expressibility: generic

Minimum number of parties: 3

Threat Model: Semi-honest

Maximum number of adversarial parties: 1

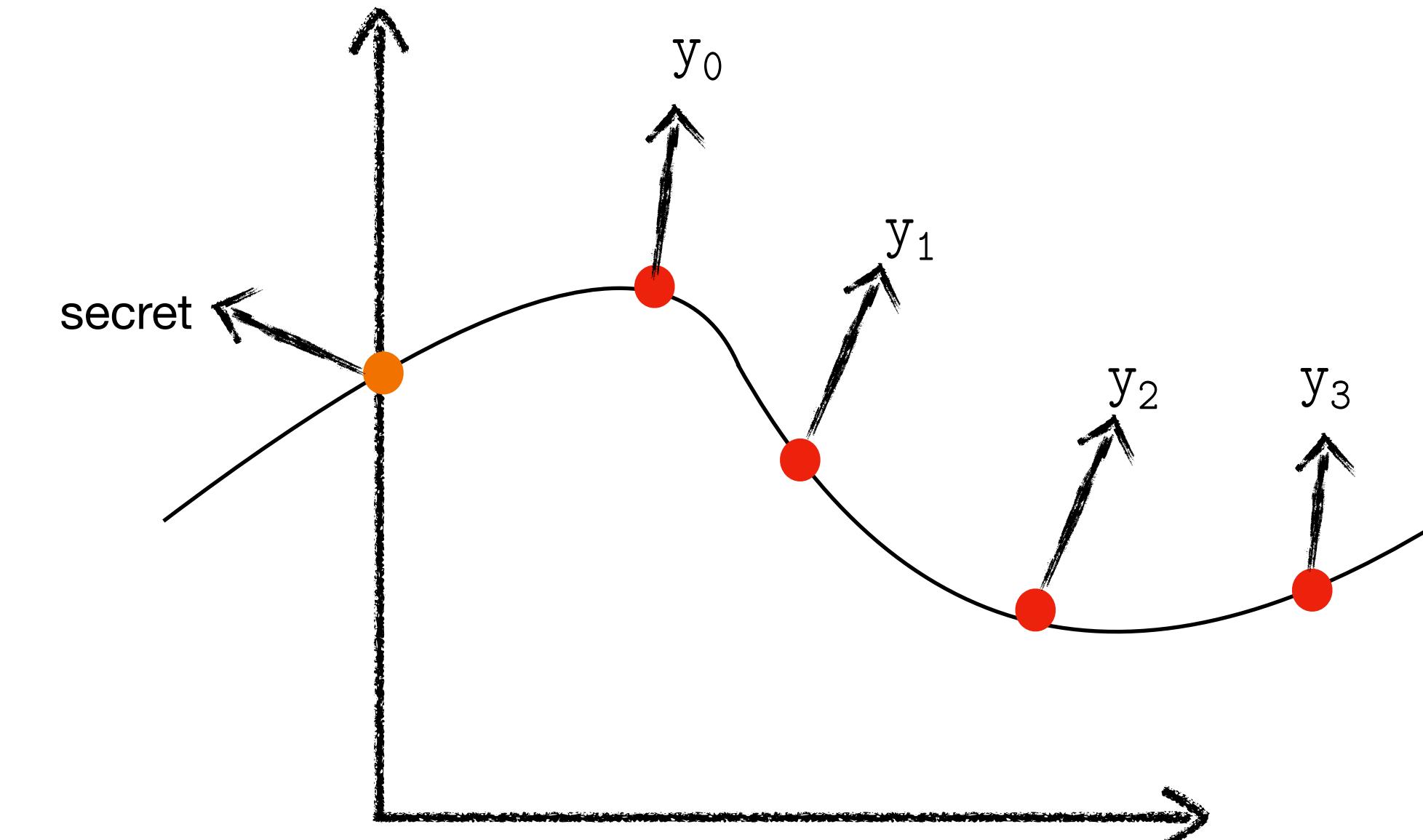
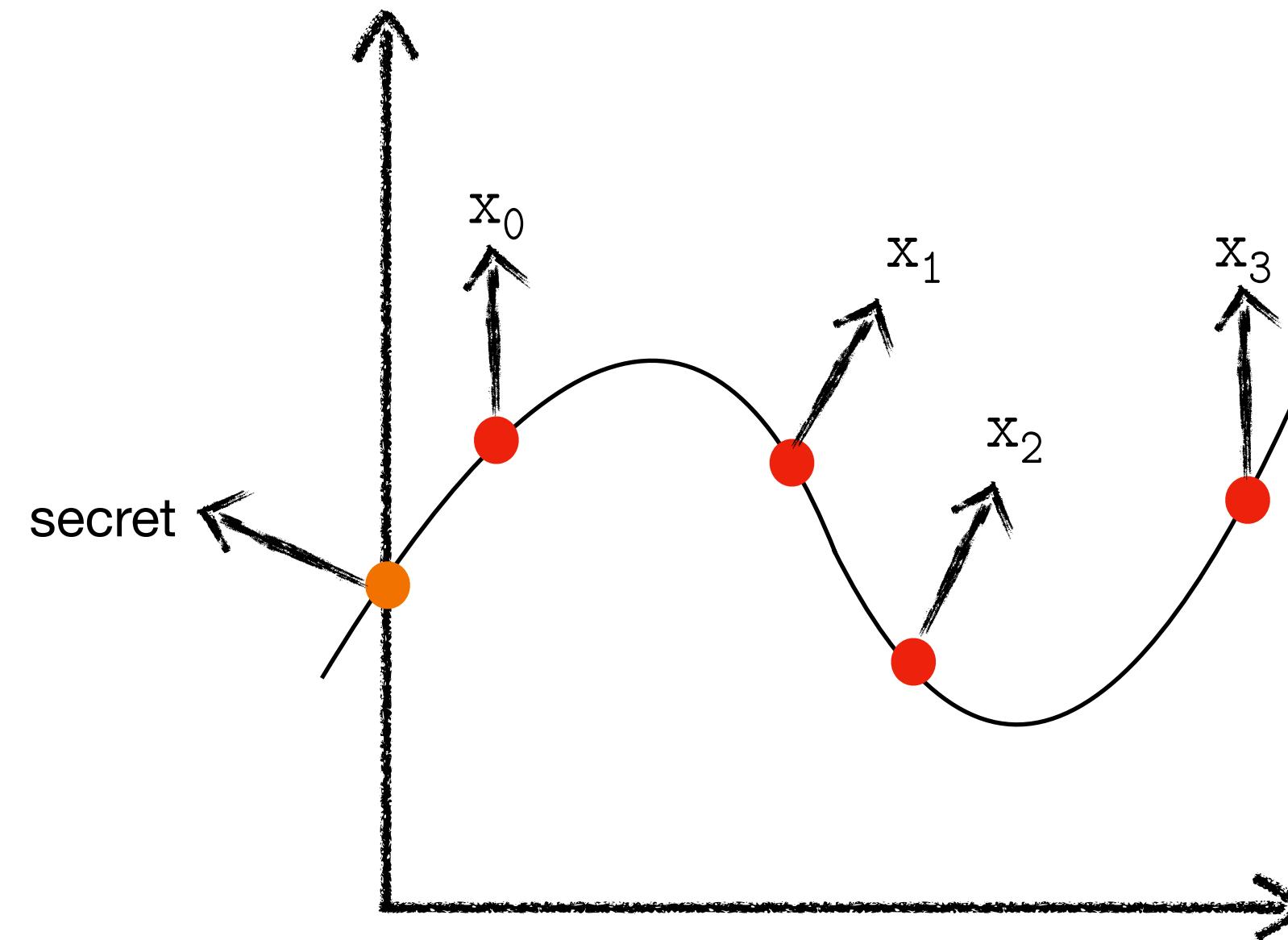
Performance ( $g$  total gates,  $m$  multiplication gates, multiplication depth  $d$ ):

Local Computation:  $\mathcal{O}(g)$

Total Communication: 3 (preprocessing) +  $m$  per party

Round Complexity: 1 preprocessing +  $d$

# Multiplication in Shamir Secret Shares



$x_0 \ y_0$



$x_2 \ y_2$



$x_1 \ y_1$



$x_3 \ y_3$

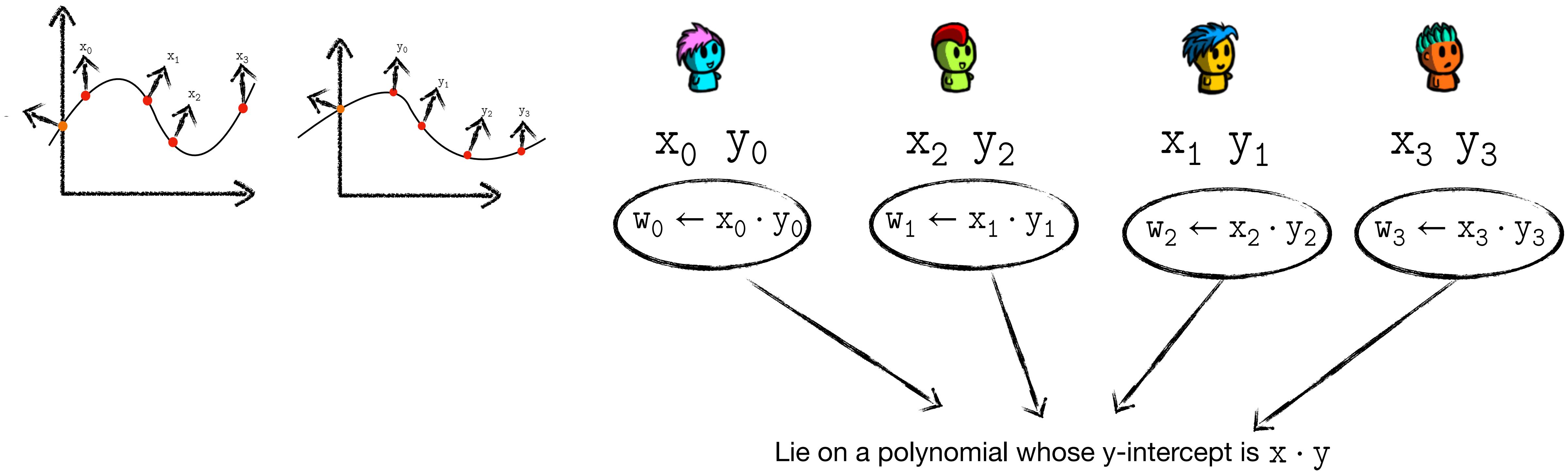
$$w_0 \leftarrow x_0 \cdot y_0$$

$$w_1 \leftarrow x_1 \cdot y_1$$

$$w_2 \leftarrow x_2 \cdot y_2$$

$$w_3 \leftarrow x_3 \cdot y_3$$

# Multiplication in Shamir Secret Shares



Is that good enough?

The degree of that polynomial is  $2t$  instead of  $t$ .

# Degree Reduction

Lagrange Interpolation:

We want to privately compute

$$w = \lambda_1 \cdot w_1 + \lambda_2 \cdot w_2 + \dots + \lambda_n \cdot w_n$$

The diagram illustrates the computation of a weighted sum. It shows three circles, each containing a coefficient  $\lambda_1$ ,  $\lambda_2$ , and  $\lambda_n$ . Arrows from each circle point to the text "Public Values" located at the bottom center.

# Degree Reduction

Each party  $i$  locally computes

$$w_i \leftarrow x_i \cdot y_i$$

Shares of  $x \cdot y$  in a polynomial of degree  $2t$

Each party shares  $w_i$  among the  $n$  parties with the correct  $t$

$w_{i,j} \rightarrow$  share of  $w_i$  held by party  $j$

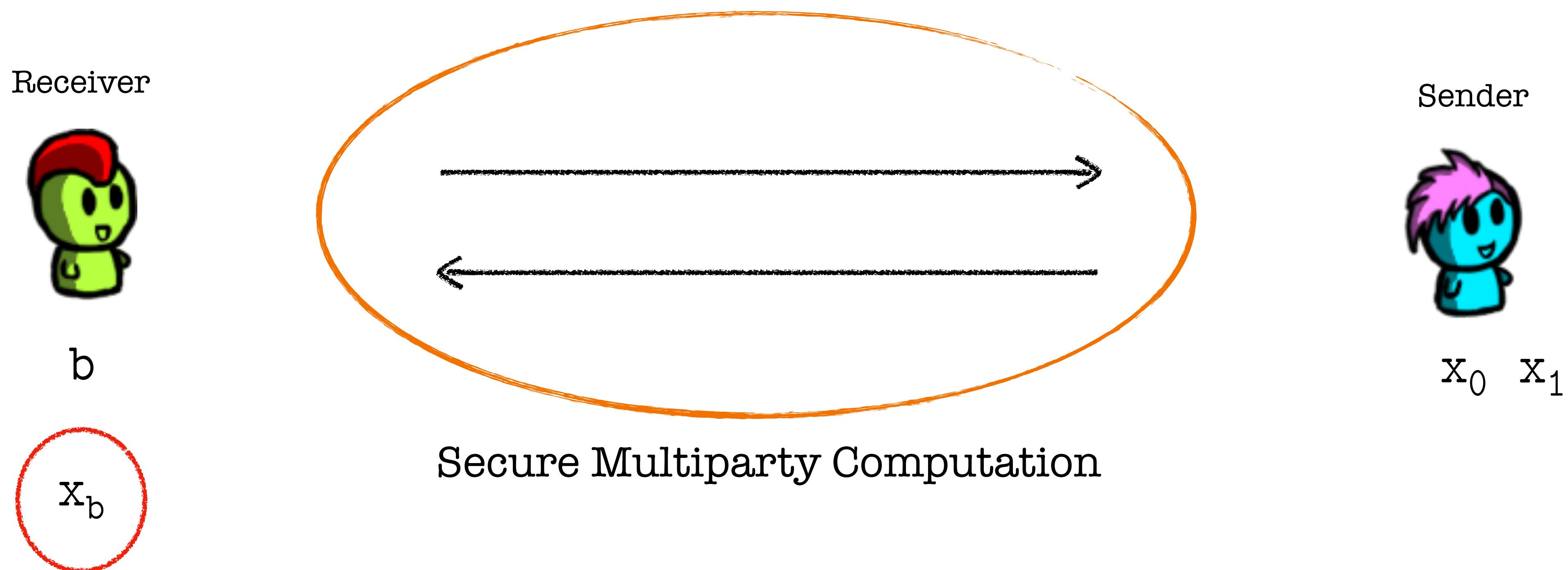
$$z_j \leftarrow \lambda_1 \cdot w_{1,j} + \lambda_2 \cdot w_{2,j} + \dots + \lambda_n \cdot w_{n,j}$$

# Degree Reduction

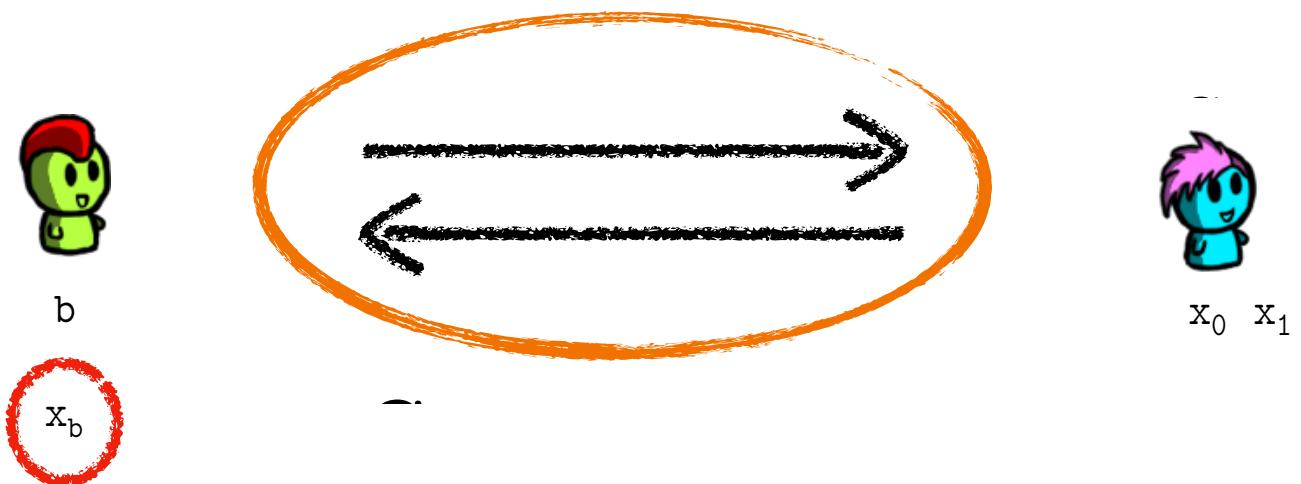
What is the minimum number of parties needed to execute this protocol?

We must have enough parties to reconstruct the intercept of the polynomial with degree  $2t$

# Oblivious Transfer



# Oblivious Transfer



What is a simple semi-honest protocol to execute Oblivious Transfer?

Bob samples a *public key* and *private key* pair  $pk, sk$

Bob samples a *random key*  $r_k$

If  $b = 1$                $(r_k, pk)$

If  $b = 0$                $(pk, r_k)$

Alice receives     $(k_0, k_1)$

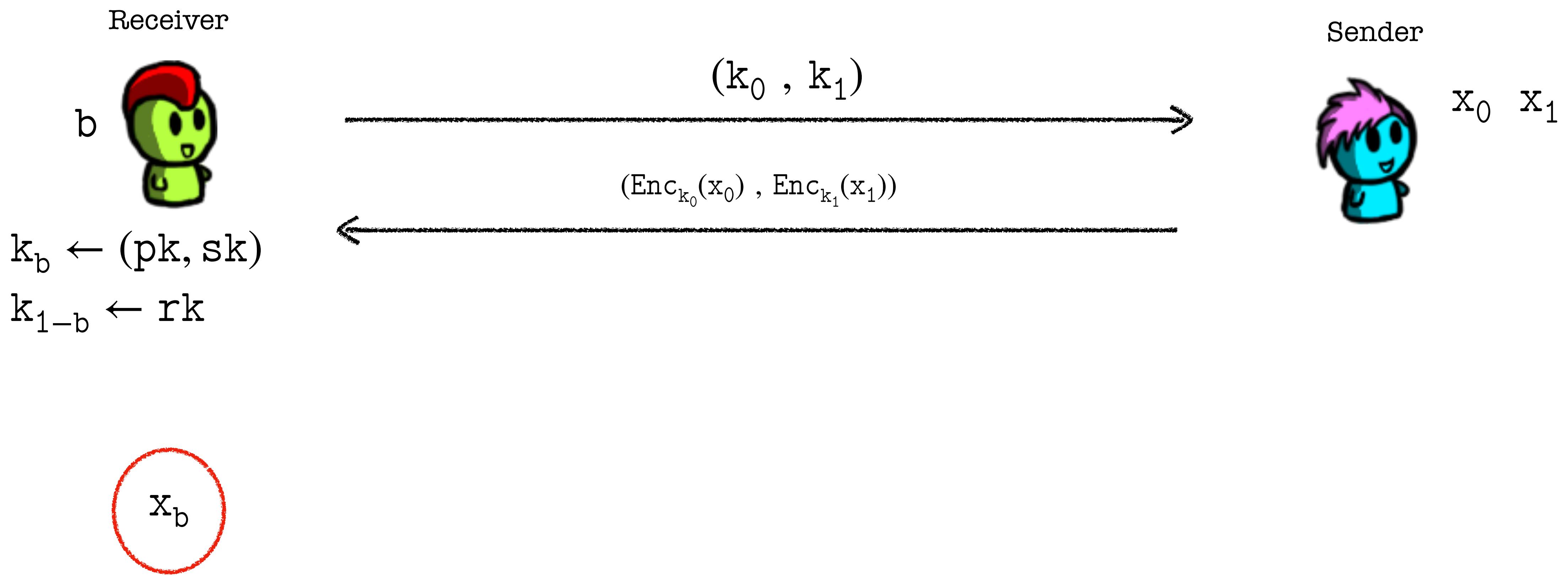
Alice sends back     $(Enc_{k_0}(x_0), Enc_{k_1}(x_1))$

Why does this work?

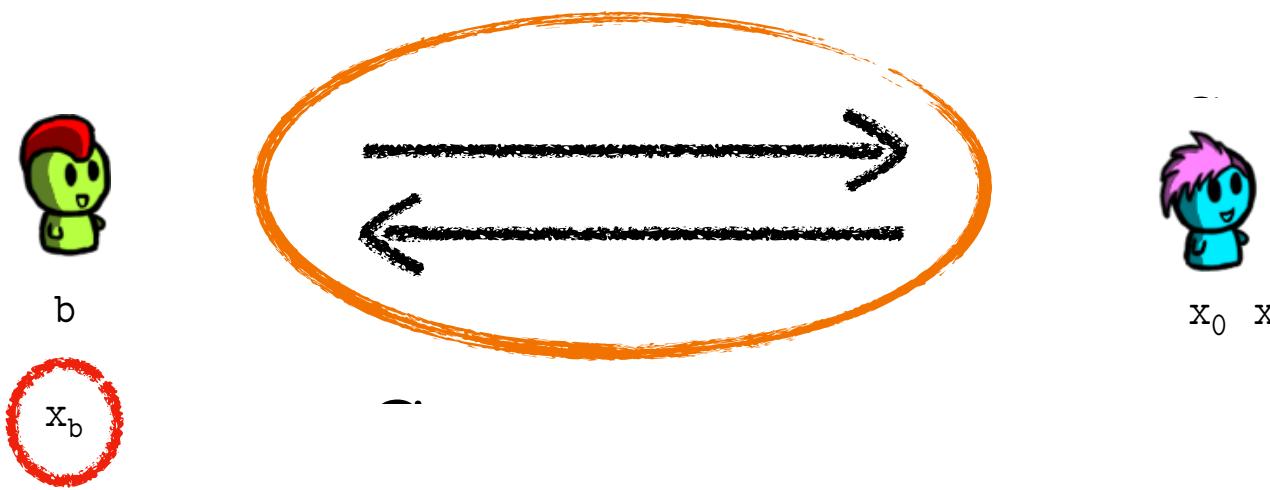
Why does this only work in a *semi-honest* setting?

Rather than sampling a random key, Bob can sample a public private key pair

# Oblivious Transfer



# Oblivious Transfer



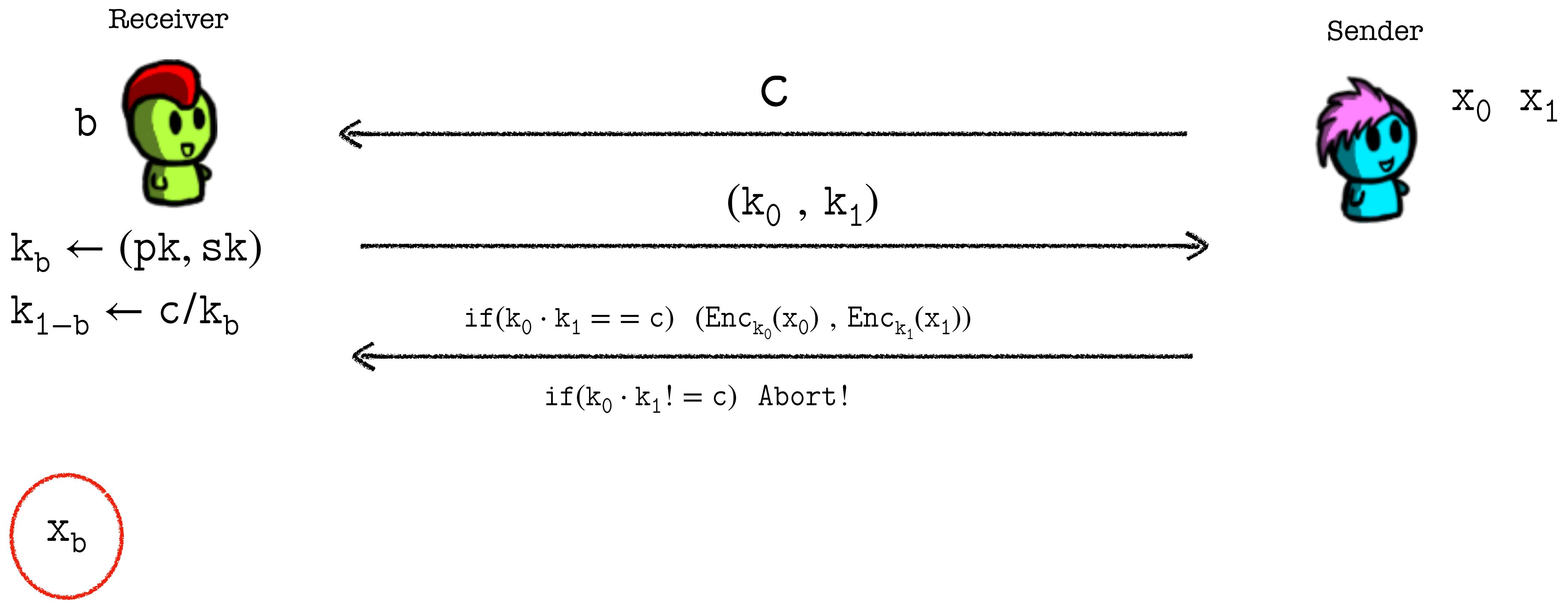
How can we make the above semi-honest protocol work under a malicious setting?

The problem with the protocol is that Bob is free to choose both keys as per his choice.

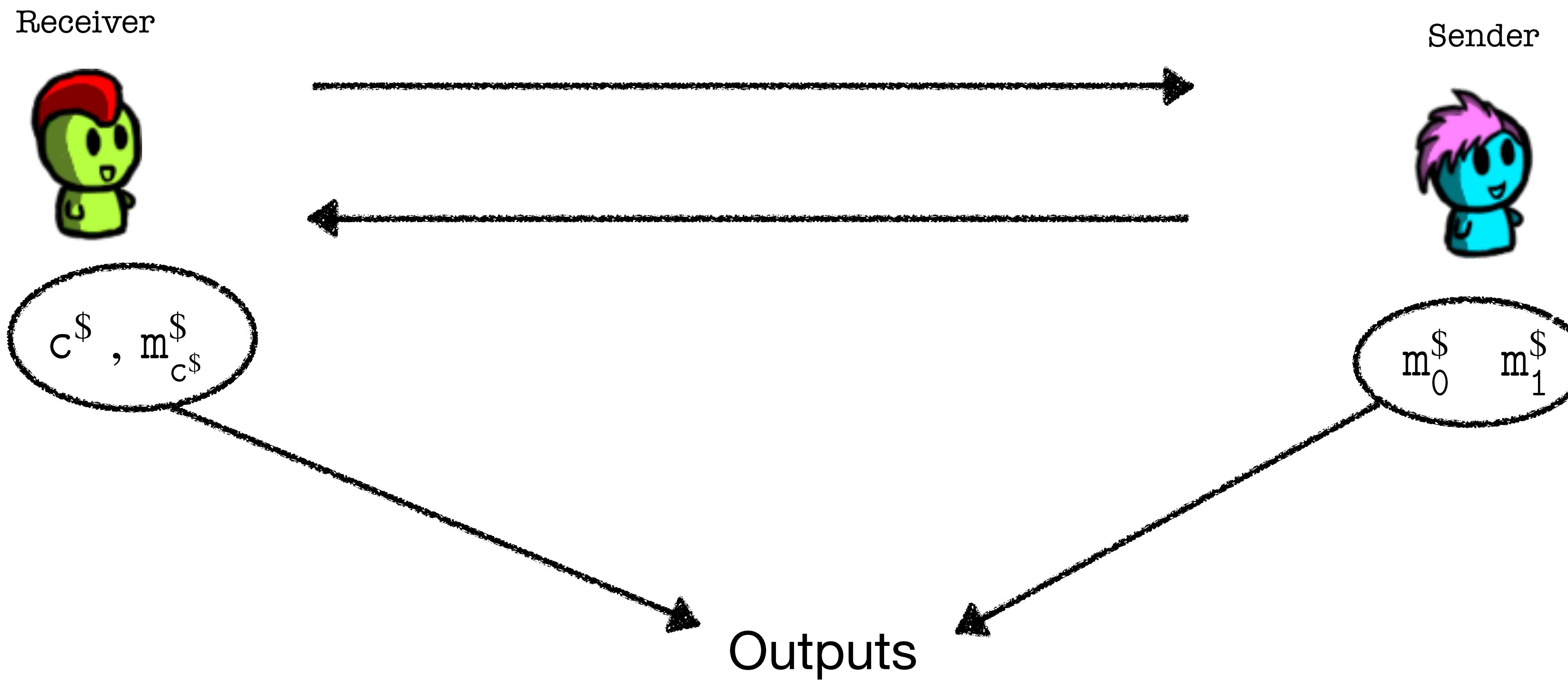


Can we somehow tie Bob's hands so that he does not do what he pleases to do?

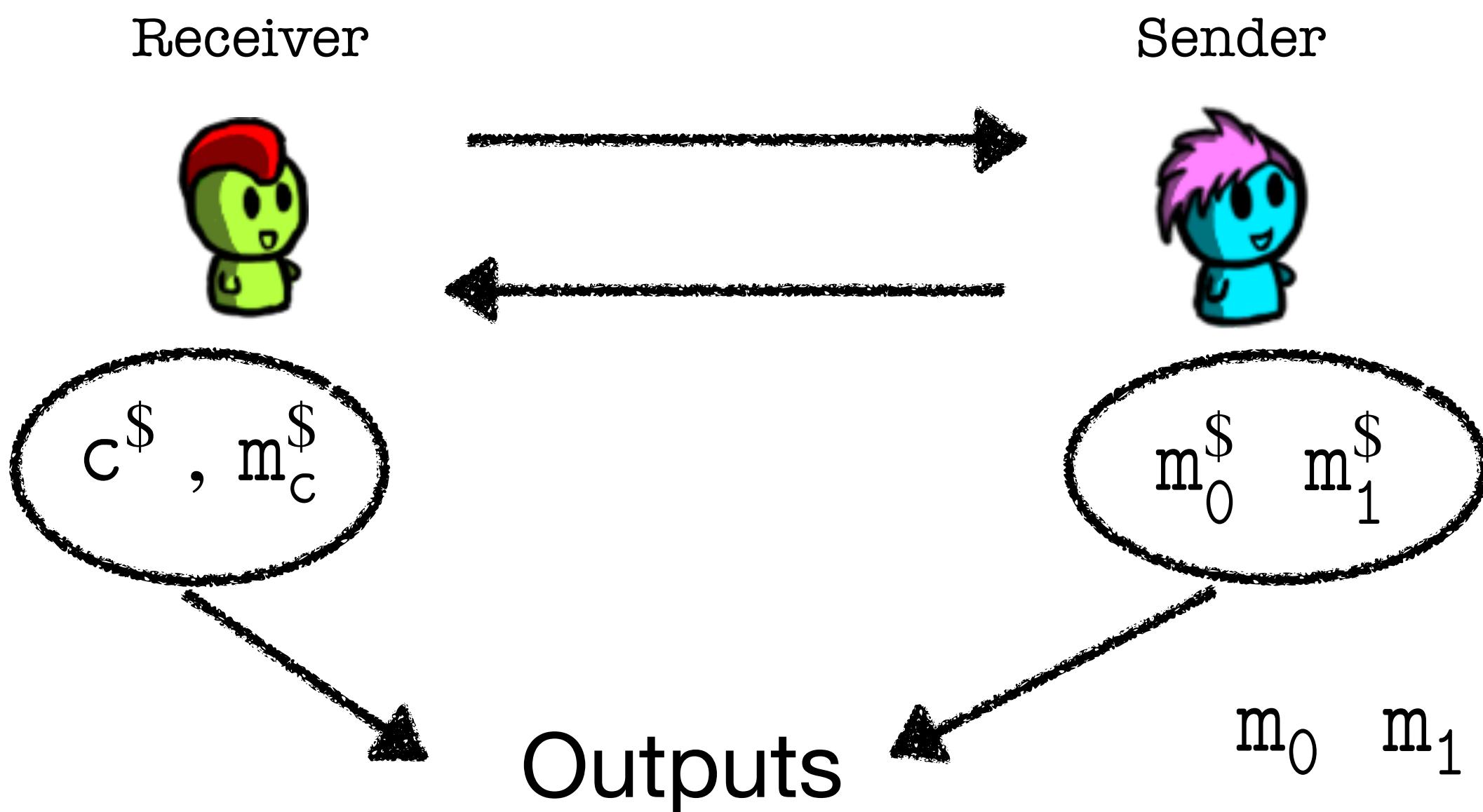
# Oblivious Transfer



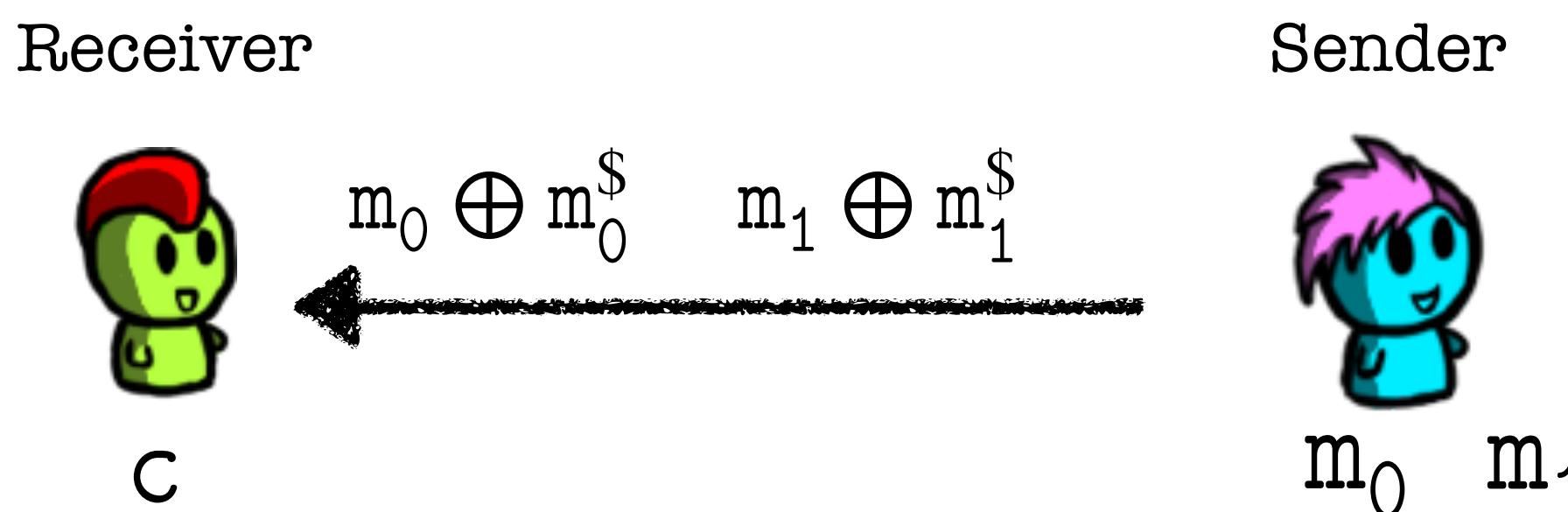
# Random Oblivious Transfer (OT)



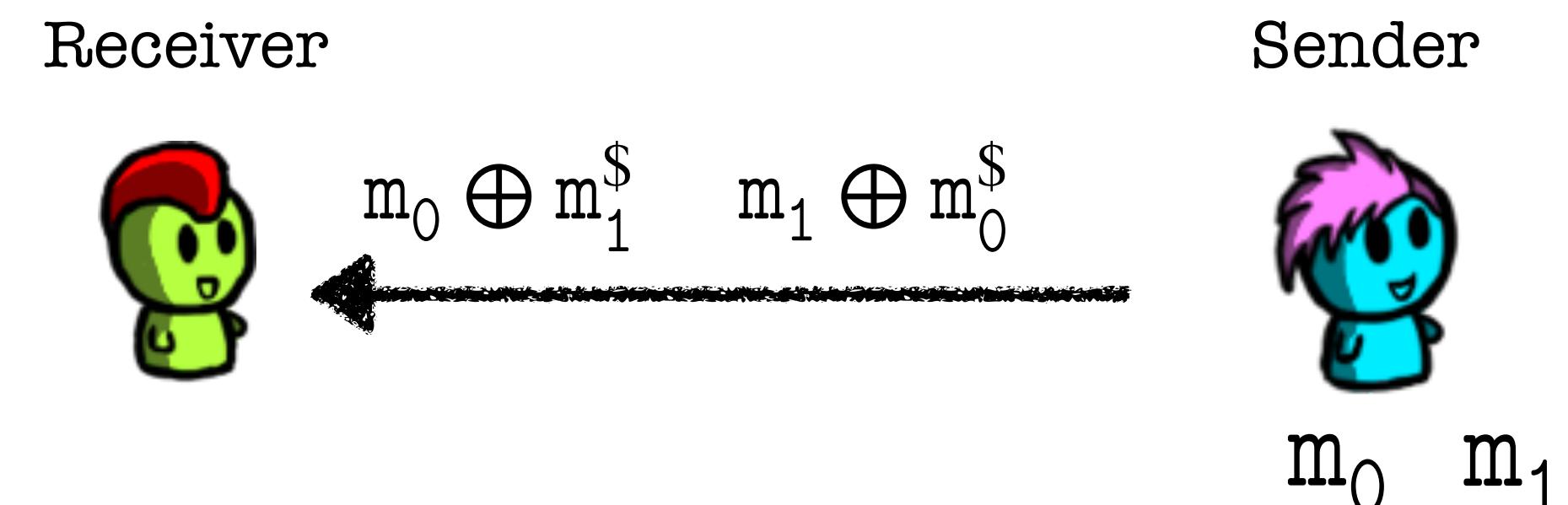
# Random Oblivious Transfer (ROT)



What is the point of this?



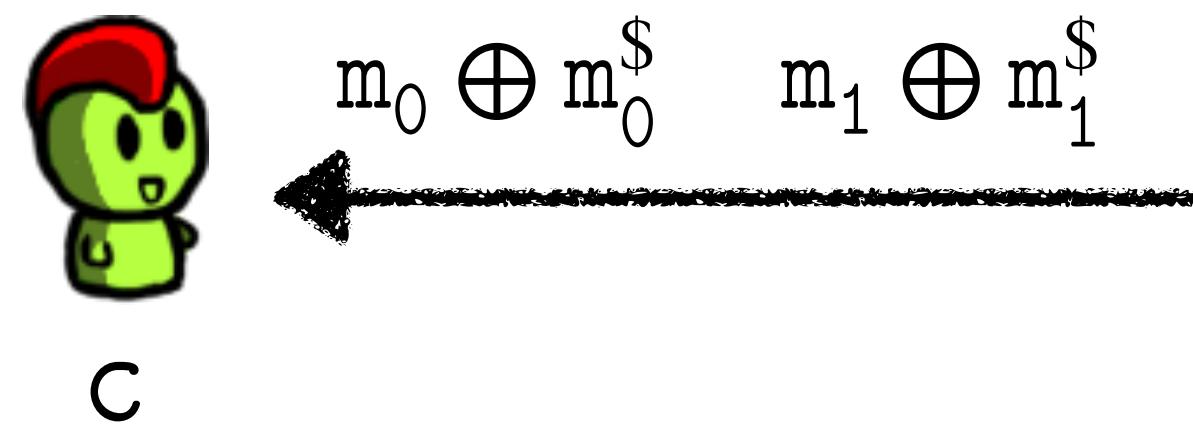
When will this work?  
If  $c = c^{\$}$



When will this work?  
If  $c = 1 - c^{\$}$

# Random Oblivious Transfer (ROT)

Receiver

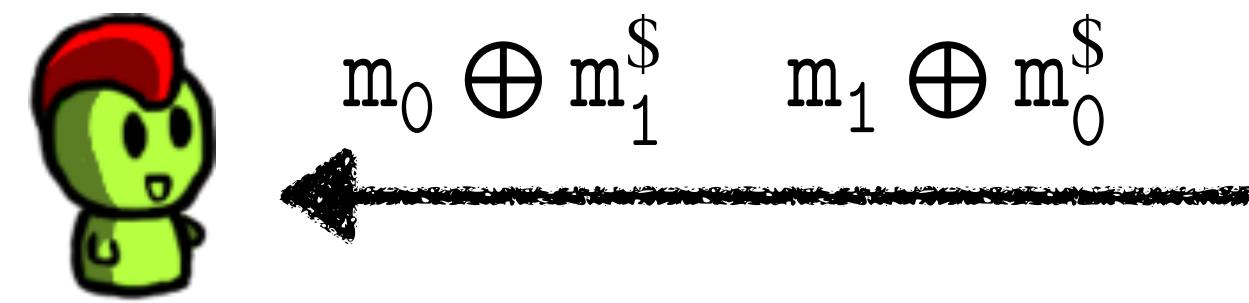


When will this work?

$$\text{If } c = c^S$$

Sender

Receiver



When will this work?

$$\text{If } c = 1 - c^S$$

Sender



How can Alice give this information to Bob?

Alice just tells it!

# OT Extension

Can we get multiple OTs for the cost of a few OTs?

# IKNP



**s = 0 1 0 0 0 1 1 0**

---

1	1	0	1	0	0	1	1
0	0	1	1	0	0	0	1
1	0	1	0	1	1	0	1
0	1	0	0	0	1	1	0
0	1	1	0	1	0	0	1
1	0	0	1	1	0	0	1
1	1	1	1	0	1	0	1
0	0	1	0	1	1	0	0

**T**

1	0	0	1	0	1	0	1
0	0	1	1	0	0	0	1
1	1	1	0	1	0	1	1
0	1	0	0	0	1	1	0
0	0	1	0	1	1	1	1
1	0	0	1	1	0	0	1
1	0	1	1	0	0	1	1
0	0	1	0	1	1	0	0

**r**

<b>1</b>	1	1	1	1	1	1	1	1
<b>0</b>	0	0	0	0	0	0	0	0
<b>1</b>	1	1	1	1	1	1	1	1
<b>0</b>	0	0	0	0	0	0	0	0
<b>1</b>	1	1	1	1	1	1	1	1
<b>0</b>	0	0	0	0	0	0	0	0
<b>1</b>	1	1	1	1	1	1	1	1
<b>0</b>	0	0	0	0	0	0	0	0
<b>1</b>	1	1	1	1	1	1	1	1
<b>1</b>	1	1	1	1	1	1	1	1
<b>1</b>	1	1	1	1	1	1	1	1
<b>0</b>	0	1	1	0	1	0	1	0



# IKNP

Bob has input  $r$  Extend to matrix and secret share as  $T$  and  $T'$

Alice chooses a random string  $s$

OT for each column: Alice obtains matrix  $Q$

Whenever  $r_i = 0$  Alice row = Bob row

$r_i = 1$  Alice row = Bob row  $\oplus s$

$$q_i = \begin{cases} t_i & r_i = 0 \\ t_i \oplus s & r_i = 1 \end{cases}$$

# IKNP

$$\begin{array}{cc} q_1 & q_1 \oplus s \\ \hline \end{array}$$

$$\begin{array}{cc} q_2 & q_2 \oplus s \\ \hline \end{array}$$

$$\begin{array}{cc} q_3 & q_3 \oplus s \\ \hline \end{array}$$

$$\begin{array}{cc} q_4 & q_4 \oplus s \\ \hline \end{array}$$

$$\begin{array}{c} t_1 \\ \hline \end{array}$$

$$\begin{array}{c} t_2 \\ \hline \end{array}$$

$$\begin{array}{c} t_3 \\ \hline \end{array}$$

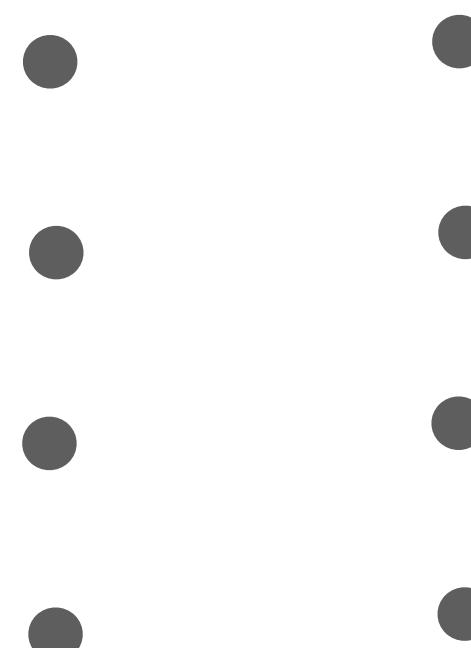
$$\begin{array}{c} t_4 \\ \hline \end{array}$$

$$q_i =$$

$$t_i$$

$$t_i \oplus s$$

$$\left. \begin{array}{l} r_i = 0 \\ r_i = 1 \end{array} \right\}$$



# IKNP

$$\begin{array}{cc}
 t_1 \oplus s & t_1 \\
 \hline
 t_2 & t_2 \oplus s \\
 \hline
 t_3 \oplus s & t_3 \\
 \hline
 t_4 & t_4 \oplus s
 \end{array}$$

⋮

$$\begin{array}{cc}
 t_1 & r_1 = 1 \\
 \hline
 t_2 & r_2 = 0 \\
 \hline
 t_3 & r_3 = 1 \\
 \hline
 t_4 & r_4 = 0
 \end{array}$$

⋮

$q_i =$

$$\begin{array}{ll}
 t_i & r_i = 0 \\
 t_i \oplus s & r_i = 1
 \end{array}$$

$$\left. \begin{array}{ll}
 r_i = 0 \\
 r_i = 1
 \end{array} \right\}$$

# IKNP

$$q_i = \begin{cases} t_i & r_i = 0 \\ t_i \oplus s & r_i = 1 \end{cases}$$

$t_1 \oplus s \quad t_1$

$t_2 \quad t_2 \oplus s$

$t_3 \oplus s \quad t_3$

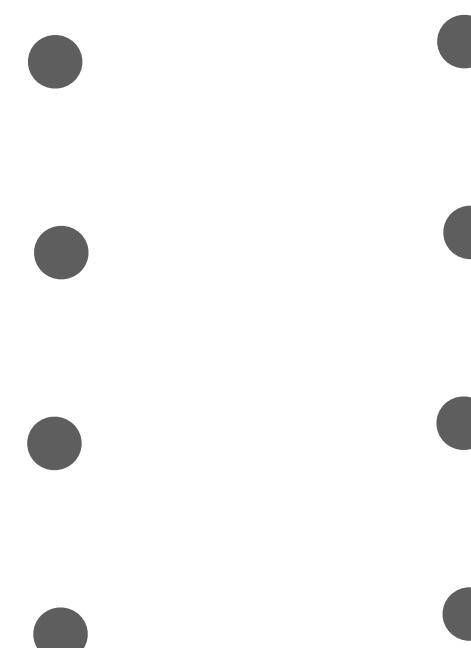
$t_4 \quad t_4 \oplus s$

$t_1 \quad r_1 = 1$

$t_2 \quad r_2 = 0$

$t_3 \quad r_3 = 1$

$t_4 \quad r_4 = 0$



For every  $i$  Bob gets  $t_i$

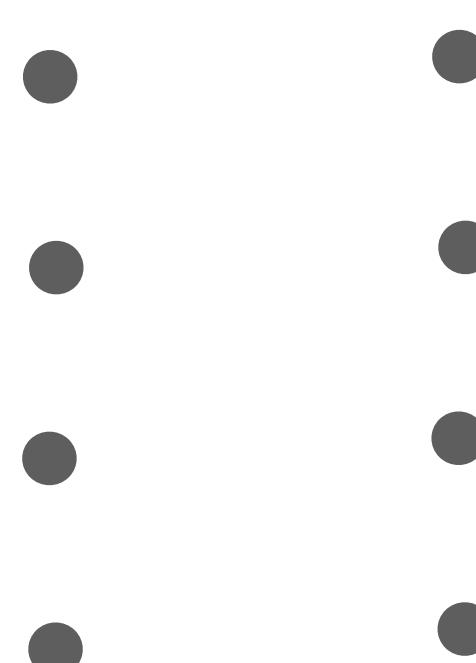
Alice can compute  $q_i \quad q_i \oplus s$

Bob knows exactly one of these two values

Almost an OT  
Reusing  $s$  leads to correlations

# IKNP

$$\begin{array}{cc}
 H(t_1 \oplus s) & H(t_1) \\
 \hline
 H(t_2) & H(t_2 \oplus s) \\
 \hline
 H(t_3 \oplus s) & H(t_3) \\
 \hline
 H(t_4) & H(t_4 \oplus s)
 \end{array}$$



$$\begin{array}{cc}
 H(t_1) & r_1 = 1 \\
 \hline
 H(t_2) & r_2 = 0 \\
 \hline
 H(t_3) & r_3 = 1 \\
 \hline
 H(t_4) & r_4 = 0
 \end{array}$$

⋮

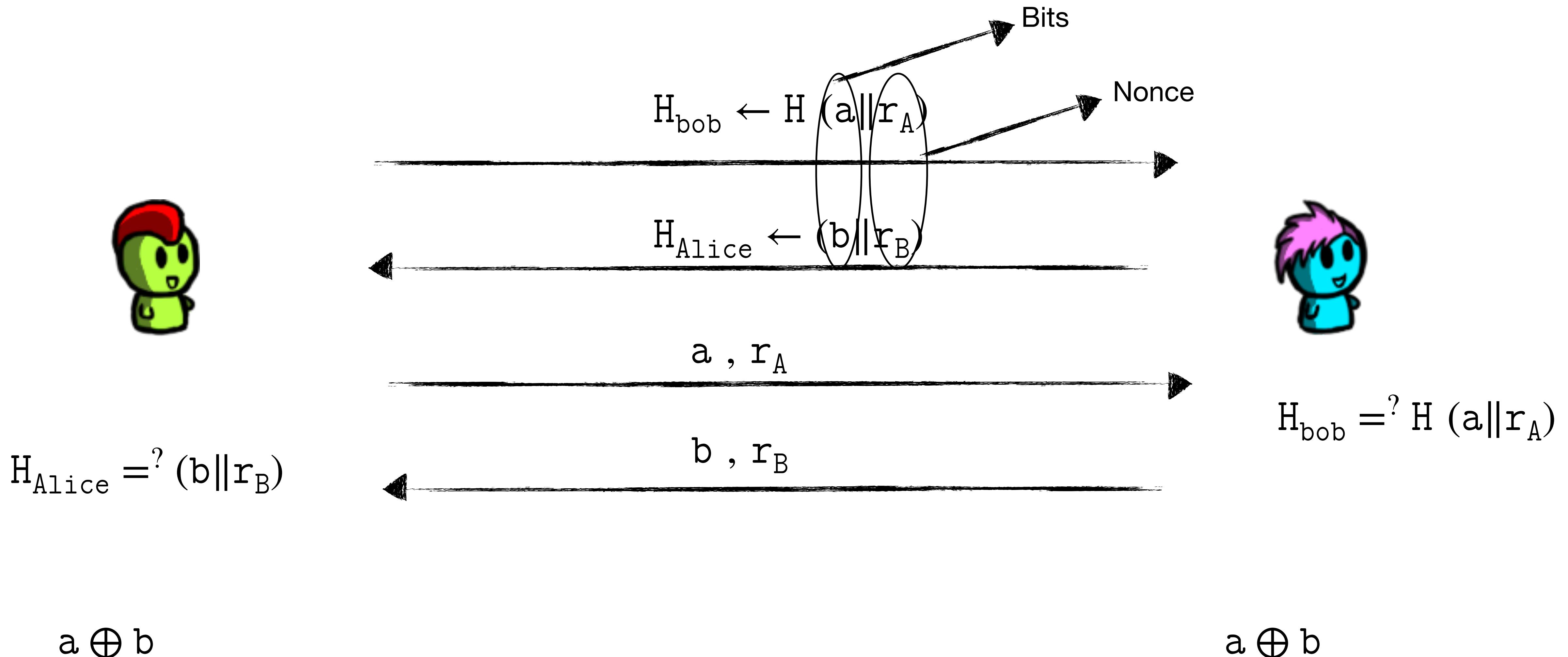
$$q_i = \begin{cases} t_i & r_i = 0 \\ t_i \oplus s & r_i = 1 \end{cases}$$

For every  $i$  Bob gets  $t_i$   
 Alice can compute  $q_i = q_i \oplus s$   
 Bob knows exactly one of these two values  
 Almost an OT  
 Reusing  $s$  leads to correlations

IKNP

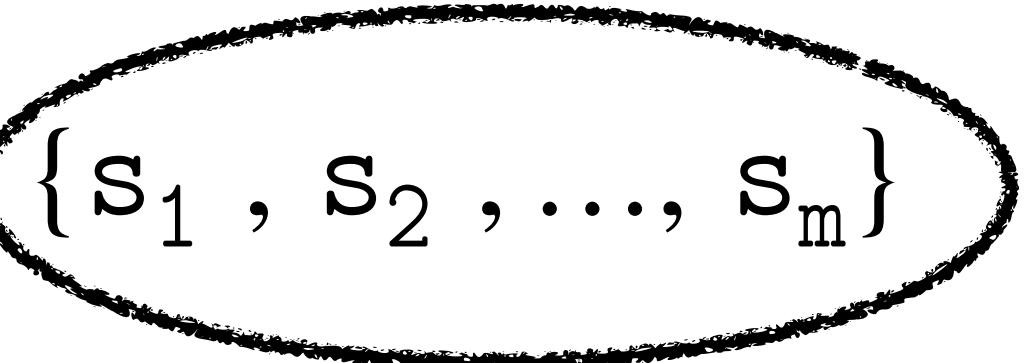
Does this work in a malicious setting?

# Tossing Coin Over Phone (Blum's Coin Toss)



# Private Set Intersection

Sender



Strings held by the Sender

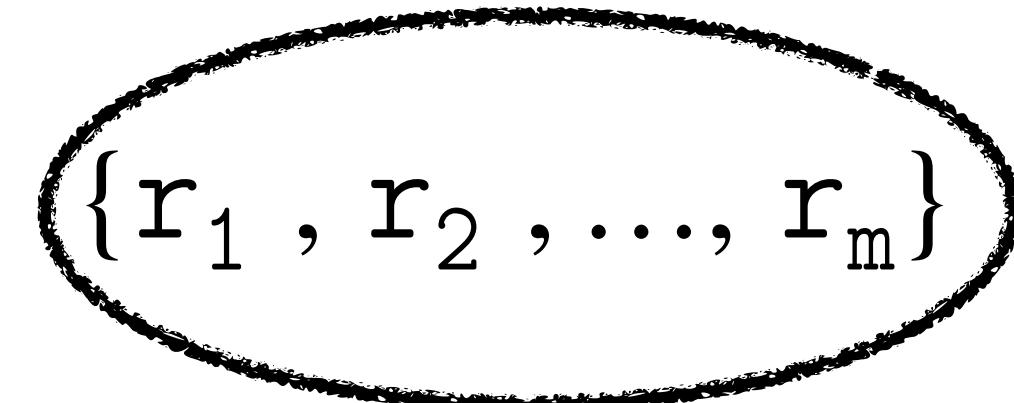
Each party has a set of elements

Goal of the receiver is to learn which elements the two parties have in common

Both parties can learn the size of each other's sets

Sender learns nothing

Receiver



Strings held by the Receiver

# Private Set Intersection

$$S = \{s_1, s_2, \dots, s_m\}$$

$$R = \{r_1, r_2, \dots, r_m\}$$



$$\{H(s_1), H(s_2), \dots, H(s_m)\}$$



Does this work?

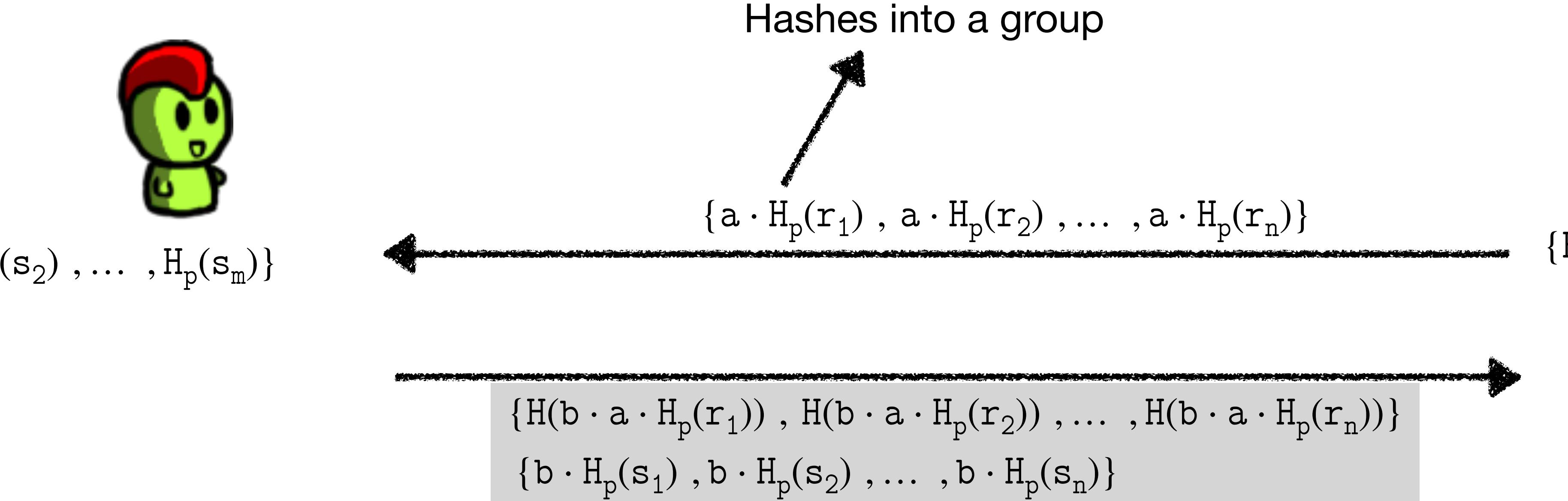
# Private Set Intersection

$$S = \{s_1, s_2, \dots, s_m\}$$



$$\{H_p(s_1), H_p(s_2), \dots, H_p(s_m)\}$$

$$R = \{r_1, r_2, \dots, r_n\}$$



Why do we have them hash into a group? We can learn the ratio of the hashes if not!

What protocol does it resemble?

DH Protocol

# Garbled Circuits

One of the first methods to do MPC



Garbler



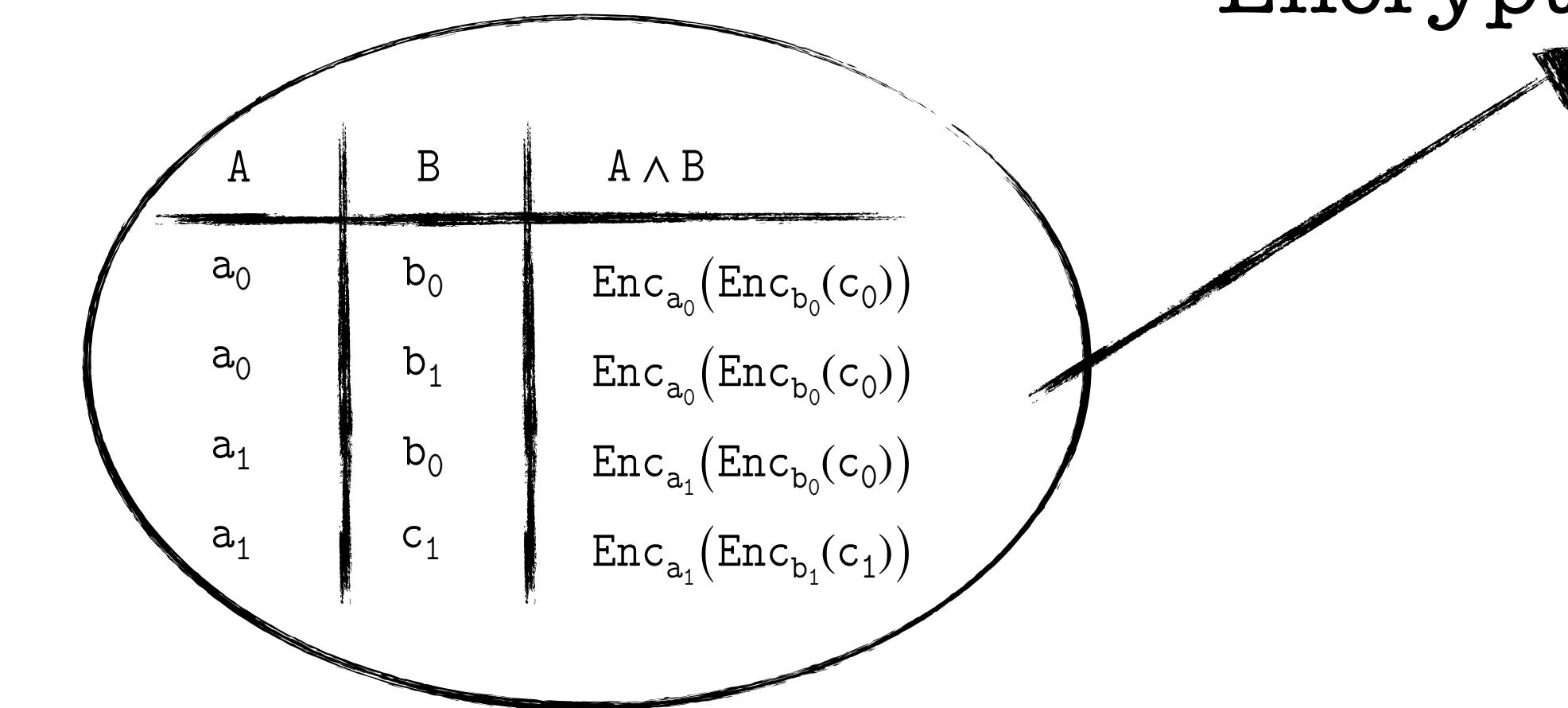
Evaluator

# Garbled Circuits

One of the first methods to do MPC

A	B	$A \wedge B$
0	0	0
0	1	0
1	0	0
1	1	1

A	B	$A \wedge B$
$a_0$	$b_0$	$c_0$
$a_0$	$b_1$	$c_0$
$a_1$	$b_0$	$c_0$
$a_1$	$c_1$	$c_1$



Garbler

Let's say we can somehow manage to give Alice the correct keys

What is still wrong with this?

The ciphertext that was decrypted will reveal the input.



Evaluator

# Garbled Circuits

One of the first methods to do MPC

A	B	$A \wedge B$
0	0	0
0	1	0
1	0	0
1	1	1

A	B	$A \wedge B$
$a_0$	$b_0$	$c_0$
$a_0$	$b_1$	$c_0$
$a_1$	$b_0$	$c_0$
$a_1$	$c_1$	$c_1$

A	B	$A \wedge B$
$a_1$	$b_0$	$\text{Enc}_{a_1}(\text{Enc}_{b_0}(c_0))$
$a_0$	$b_0$	$\text{Enc}_{a_0}(\text{Enc}_{b_0}(c_0))$
$a_1$	$c_1$	$\text{Enc}_{a_1}(\text{Enc}_{b_1}(c_1))$
$a_0$	$b_1$	$\text{Enc}_{a_0}(\text{Enc}_{b_0}(c_0))$

Permuted  
Encrypted Circuit



Garbler

How can we send the correct keys to Alice!?

Start with Bob: How can we get Alice Bob's key?

Simply tell the key



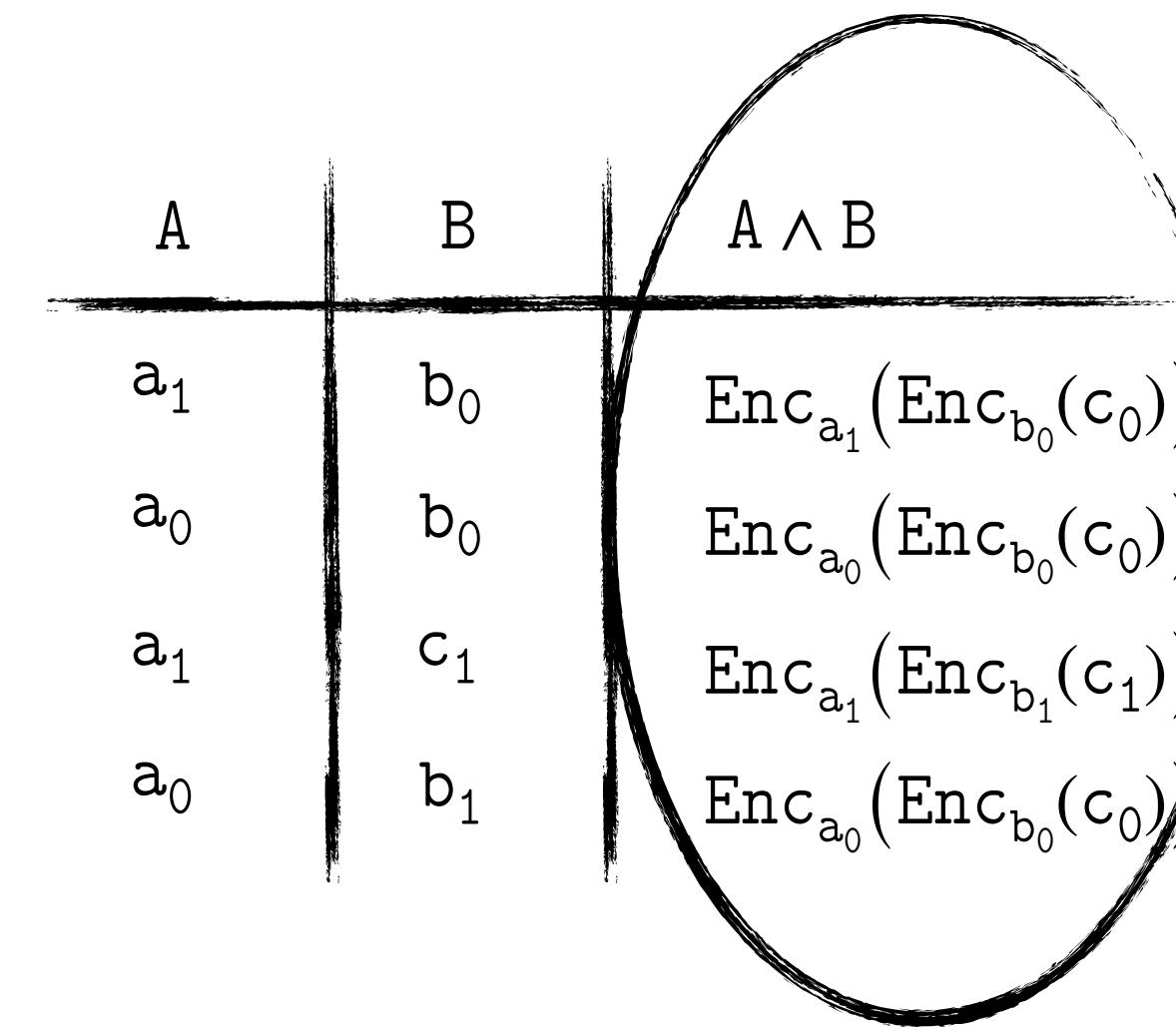
Evaluator

# Garbled Circuits

One of the first methods to do MPC

A	B	$A \wedge B$
0	0	0
0	1	0
1	0	0
1	1	1

A	B	$A \wedge B$
$a_0$	$b_0$	$c_0$
$a_0$	$b_1$	$c_0$
$a_1$	$b_0$	$c_0$
$a_1$	$c_1$	$c_1$



How can we send the correct keys to Alice!?

How about Alice's Key? Oblivious Transfer

Garbler



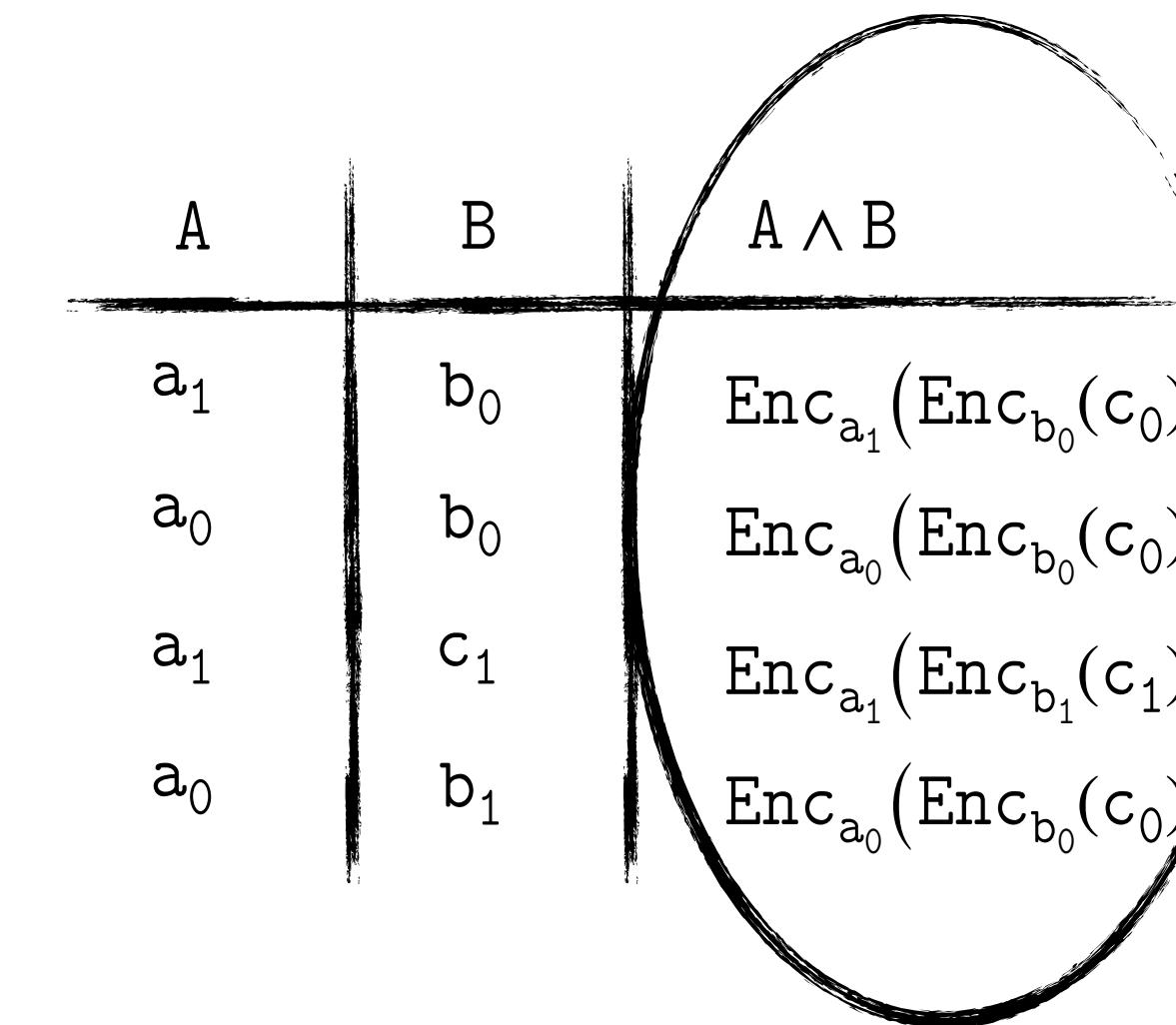
Evaluator

# Garbled Circuits

One of the first methods to do MPC

A	B	$A \wedge B$
0	0	0
0	1	0
1	0	0
1	1	1

A	B	$A \wedge B$
$a_0$	$b_0$	$c_0$
$a_0$	$b_1$	$c_0$
$a_1$	$b_0$	$c_0$
$a_1$	$c_1$	$c_1$



What kind of encryption algorithm do we need? (*what properties?*)

Garbler

We need to know if the decryption was successful or not.

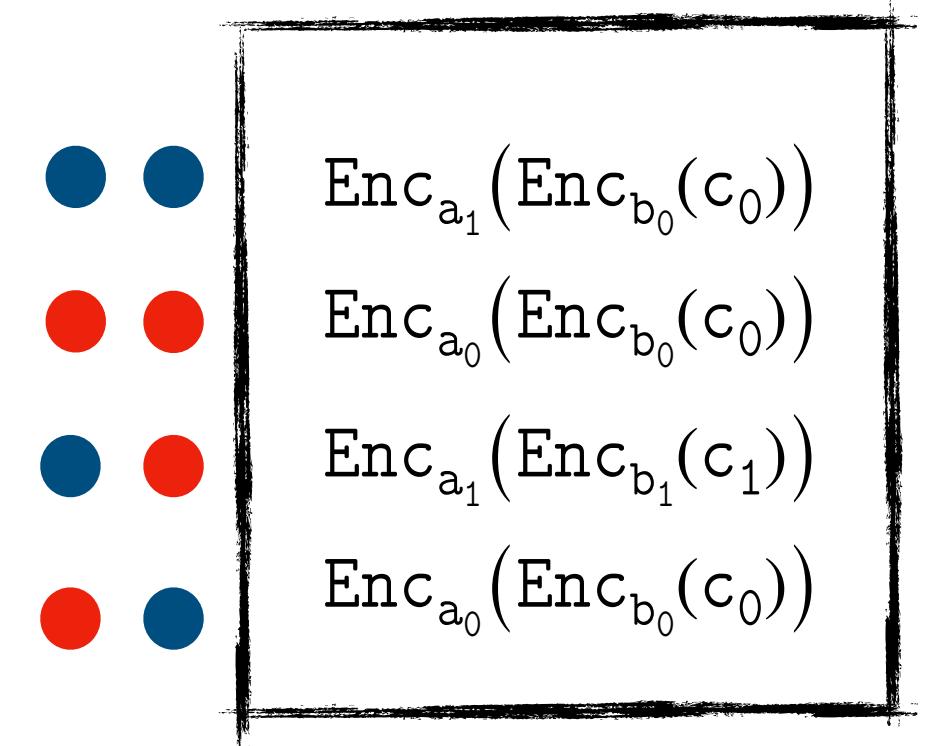
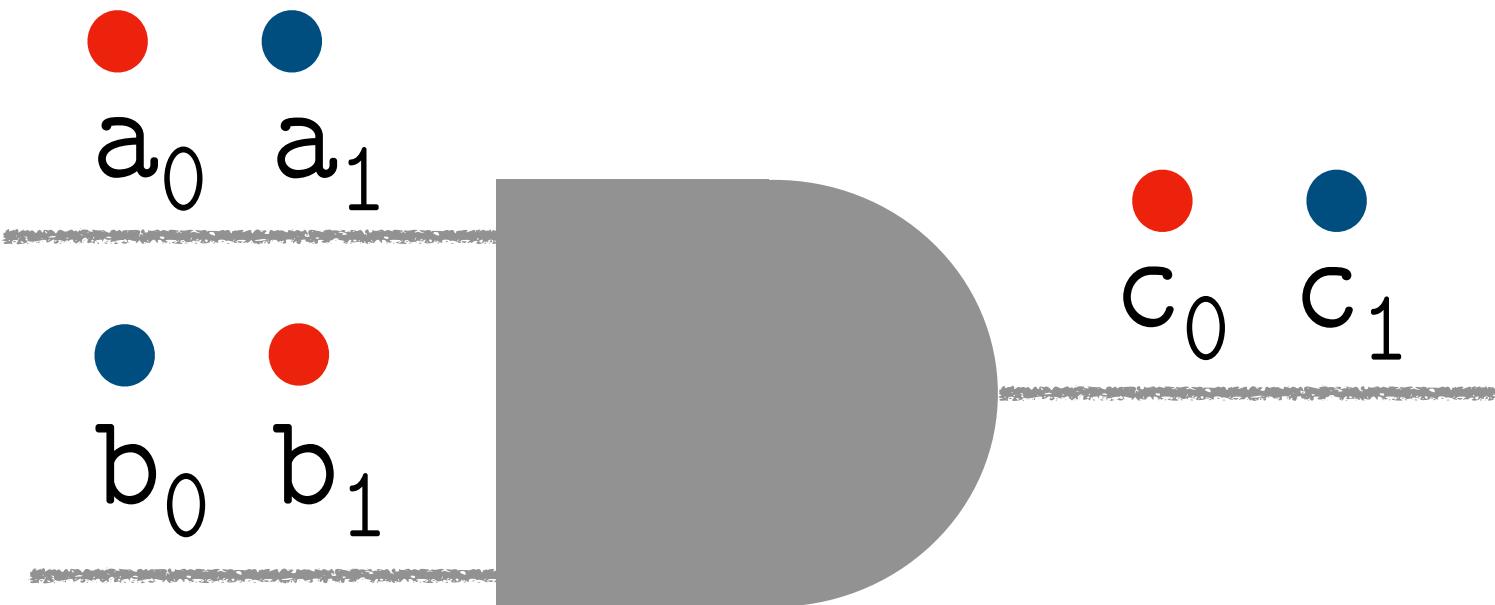


Evaluator

# Garbled Circuits: Optimization

## Point and Permute Technique

A	B	$A \wedge B$
$a_0$	$b_0$	$c_0$
$a_0$	$b_1$	$c_0$
$a_1$	$b_0$	$c_0$
$a_1$	$c_1$	$c_1$



Garbler

How do these colors help us?

Bob can simply tell Alice exactly which row to decrypt

Bob can use one-time-pads to do the encryption

Why?



Evaluator

# Garbled Circuits: Optimization

## GRR3 (Garbled Row Reduction)

A	B	$A \wedge B$
$a_0$	$b_0$	$c_0$
$a_0$	$b_1$	$c_0$
$a_1$	$b_0$	$c_0$
$a_1$	$c_1$	$c_1$

How can we send one less cipher text to the evaluator?

The Garbler can choose the wire labels such that one of the ciphers is:  $0^n$



Garbler



Evaluator

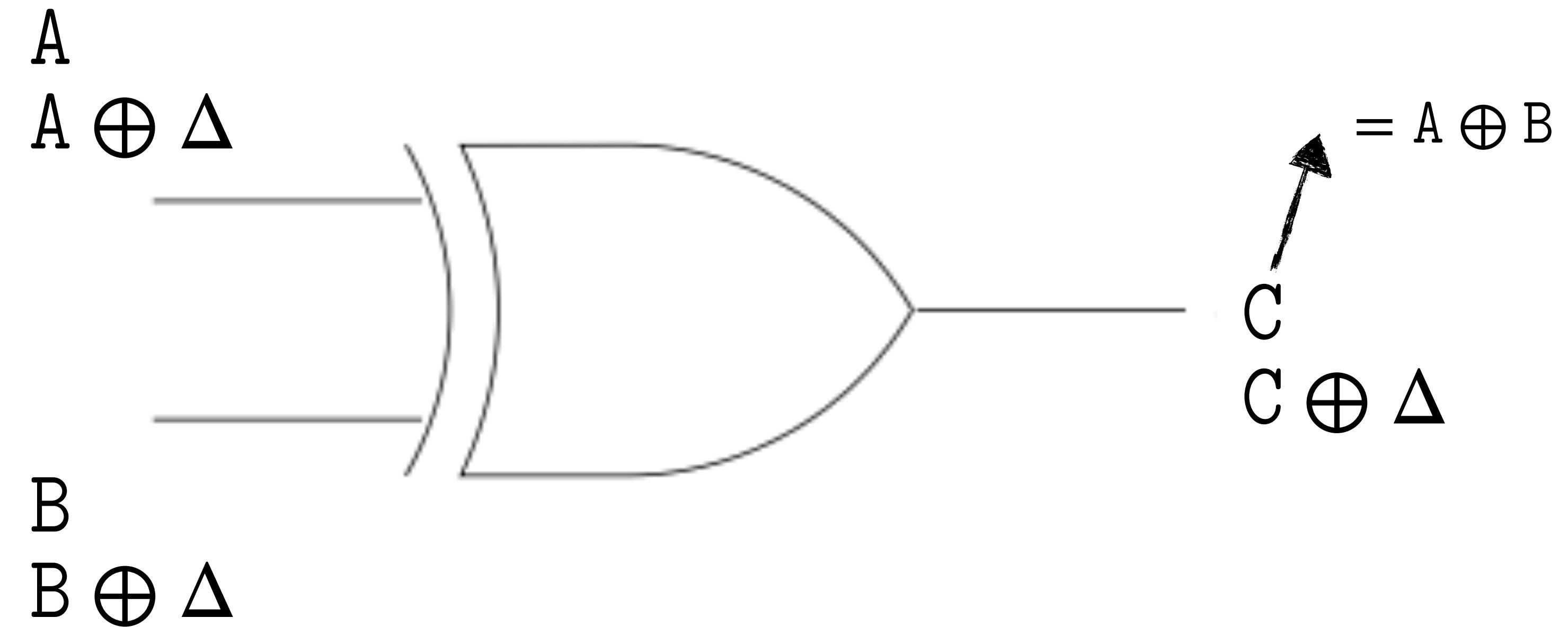
# Garbled Circuits: Optimization

Free XOR

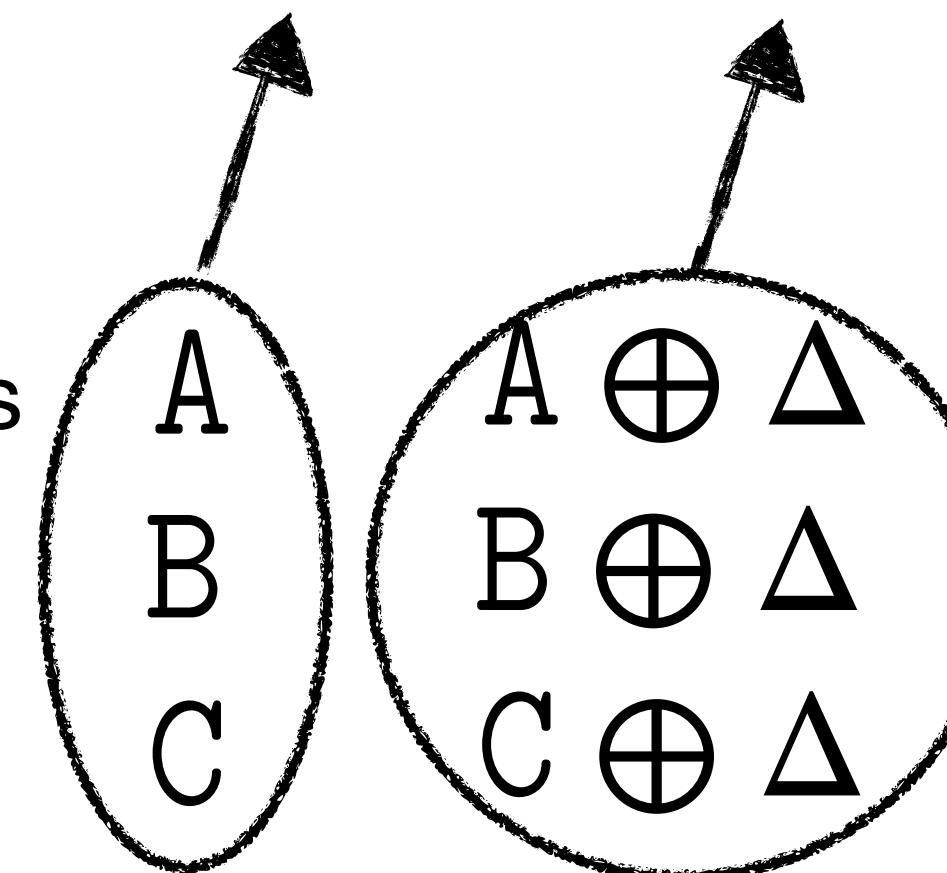
A	B	$A \wedge B$
$a_0$	$b_0$	$c_0$
$a_0$	$b_1$	$c_0$
$a_1$	$b_0$	$c_0$
$a_1$	$c_1$	$c_1$



Garbler



True Labels      False Labels



Choose an offset  $\Delta$

Set one of the input wire label as

Set the other input wire label as

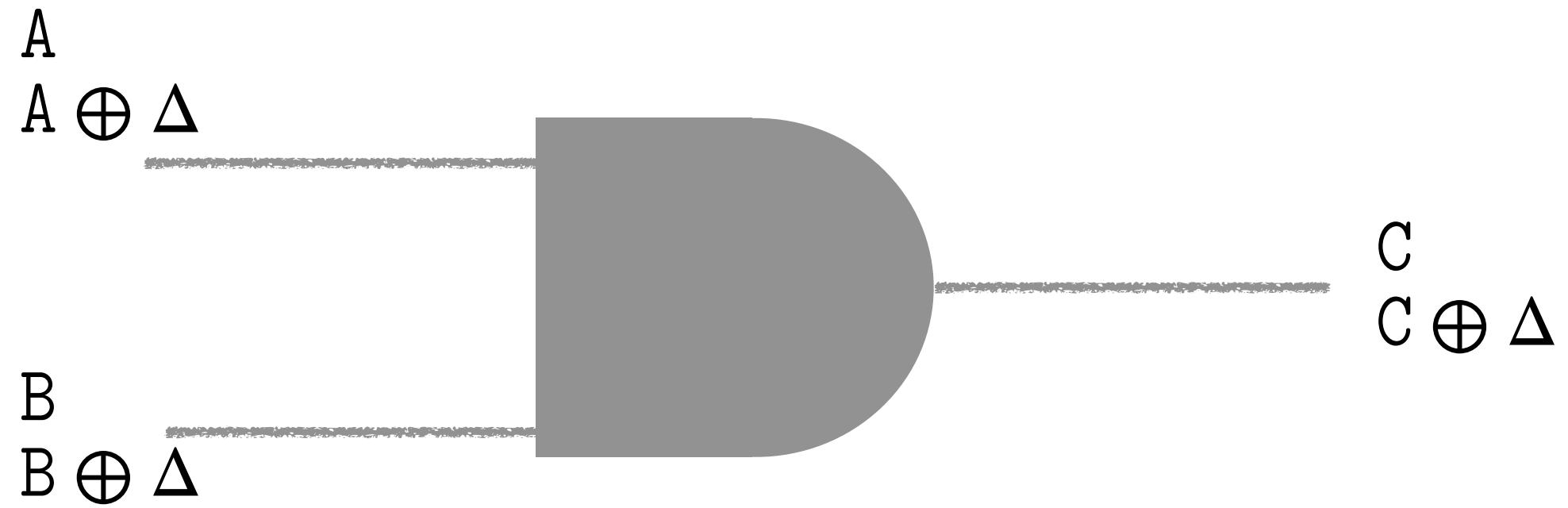
Set the output wire labels as



Evaluator

# Garbled Circuits: Optimization

## Half Gates



Suppose that garbler knew that  $a = 0$

Then the gate becomes a unary gate and always outputs 0



Garbler

Suppose that garbler knew that  $a = 1$

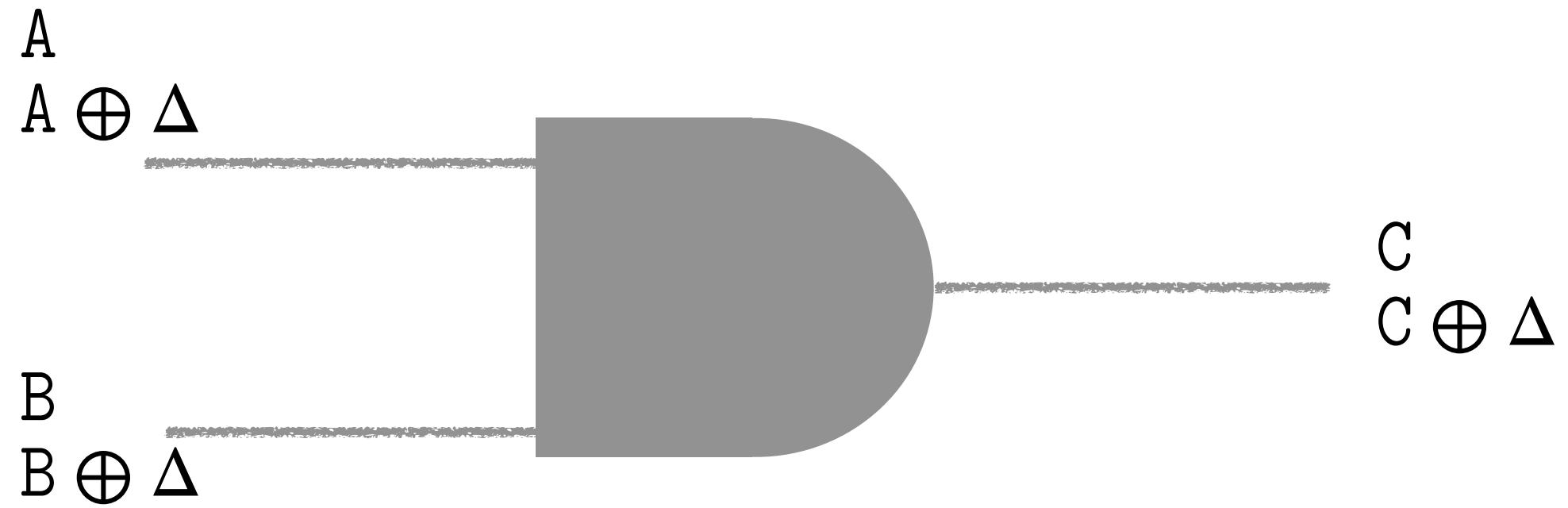
Then the gate becomes an identity gate



Evaluator

# Garbled Circuits: Optimization

## Half Gates



Suppose that garbler knew that  $a = 0$

Then the gate becomes a unary gate and always outputs 0



Garbler

Suppose that garbler knew that  $a = 1$

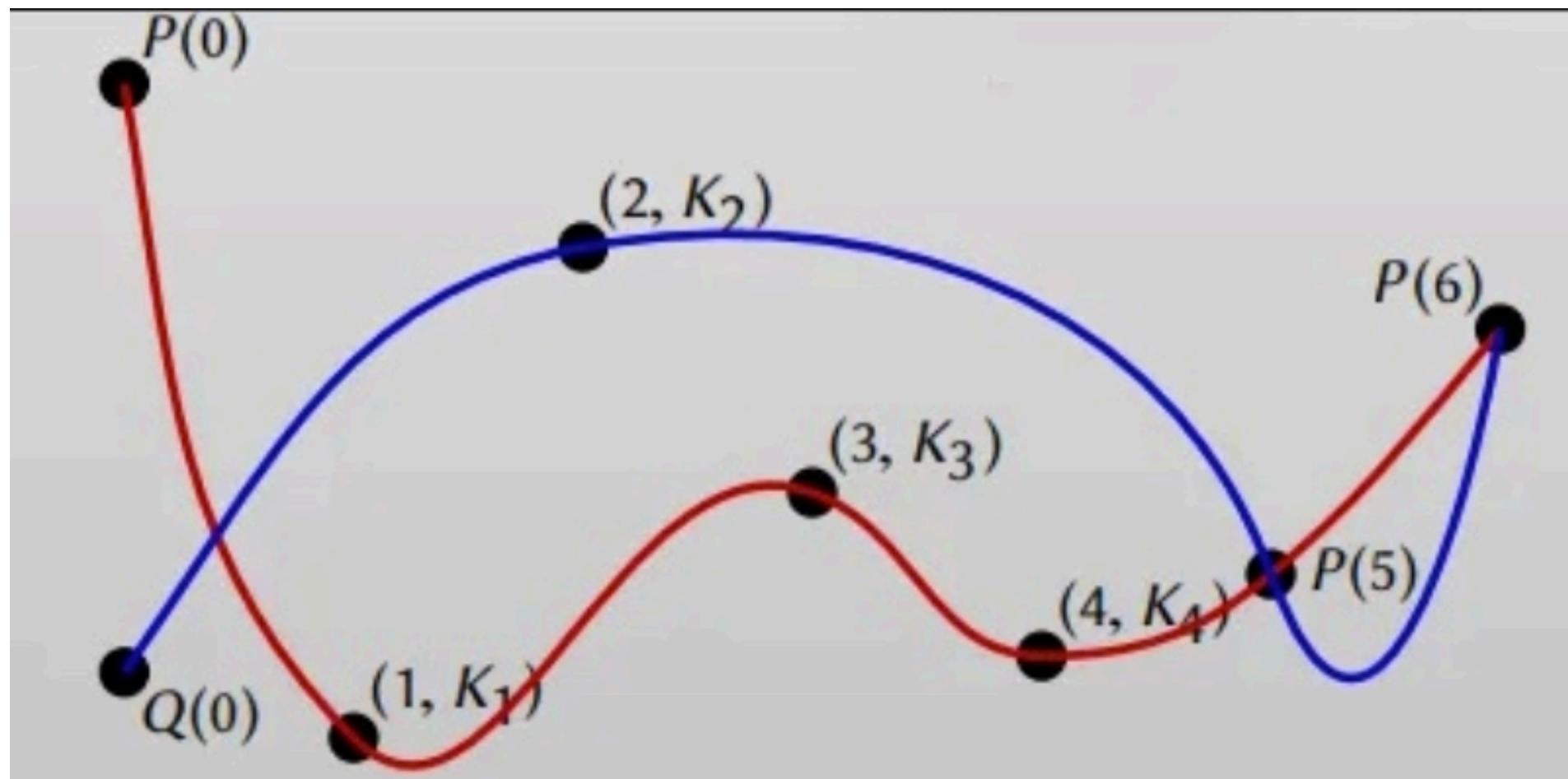
Then the gate becomes an identity gate



Evaluator

# Garbled Circuits: Optimization

GRR2



Garbler computes

$$K_1 = \text{Dec}_{a_0, b_0}(0^n)$$

$$K_2 = \text{Dec}_{a_0, b_1}(0^n)$$

$$K_3 = \text{Dec}_{a_1, b_0}(0^n)$$

$$K_4 = \text{Dec}_{a_1, b_1}(0^n)$$



Garbler

P(0) Is the first label

Q(0) Is the second label



Evaluator