

# Secure Computation

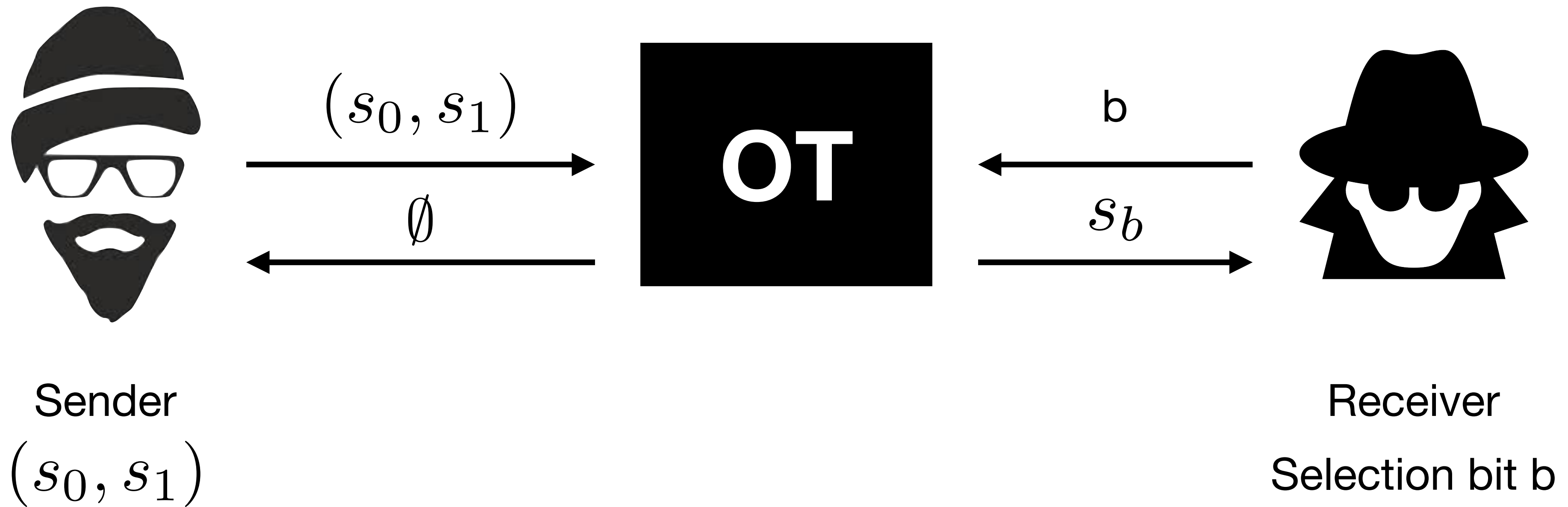
In this course, we will introduce *secure computation*, an active research area in cryptography that aims at protecting private data even when they are used in computations.

The slides for the course will be online after the course. I encourage you to take notes and try to solve the exercises which will come up during the session. If you have any question after the course, don't hesitate to mail me (address below)

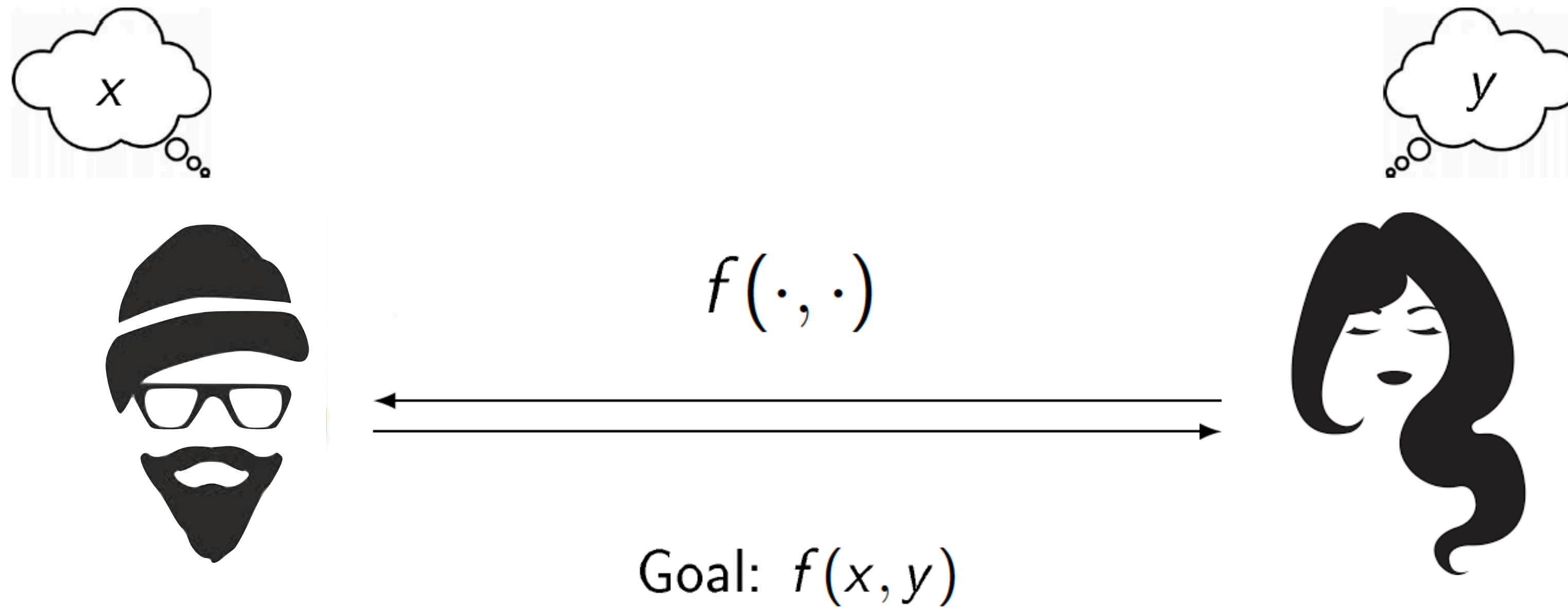
Geoffroy Couteau

[couteau@irif.fr](mailto:couteau@irif.fr)

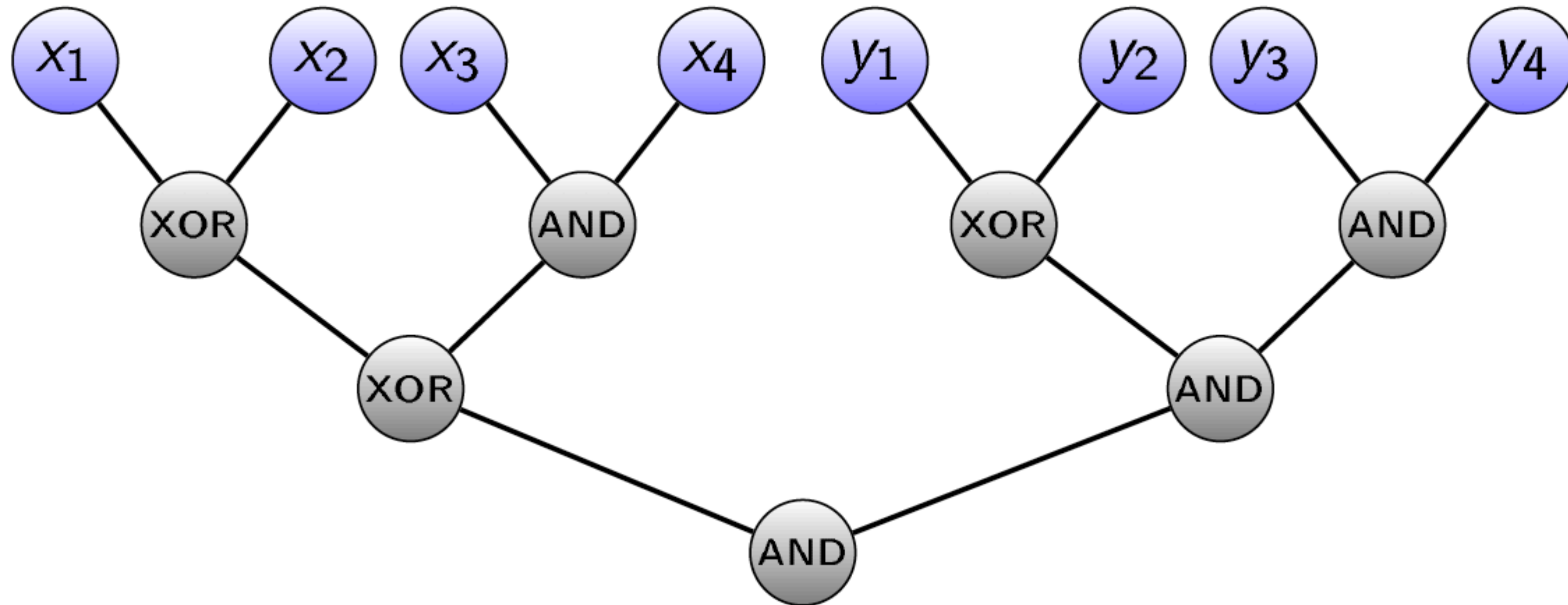
# Reminder - Oblivious Transfer



# Reminder - Two-Party Computation



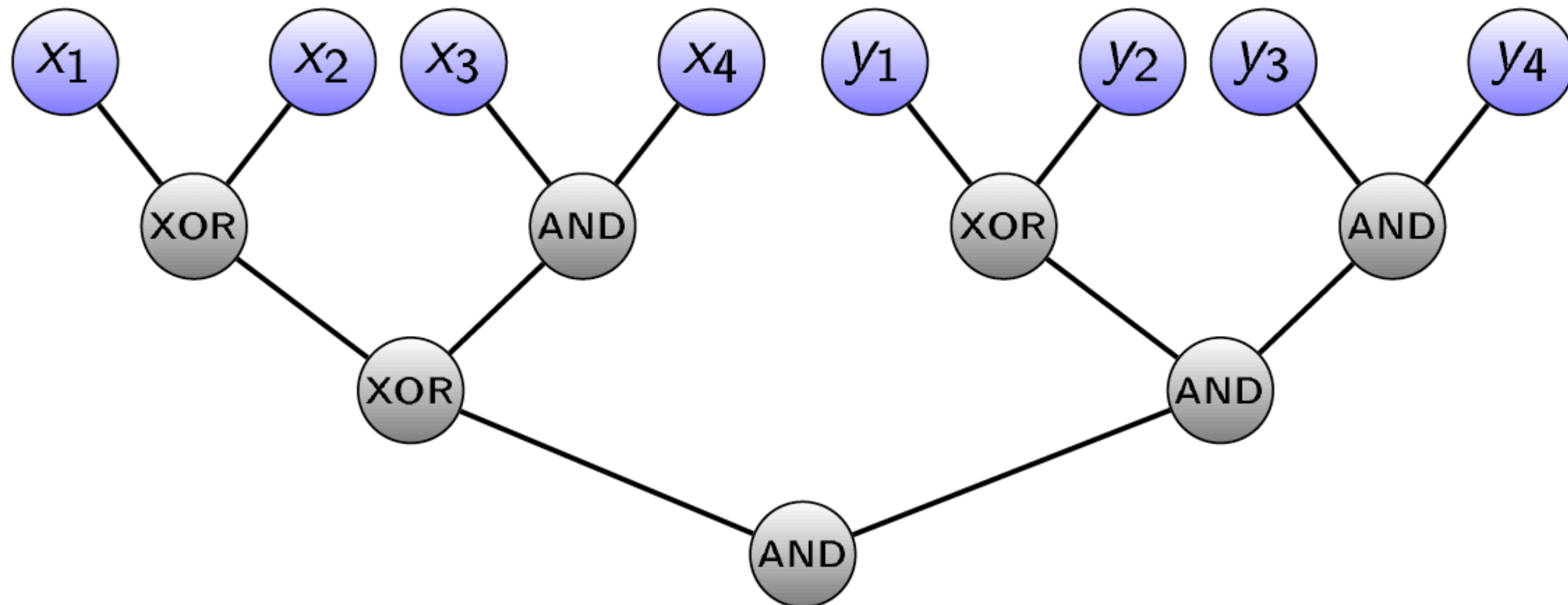
# Boolean Circuits



# Boolean Circuits

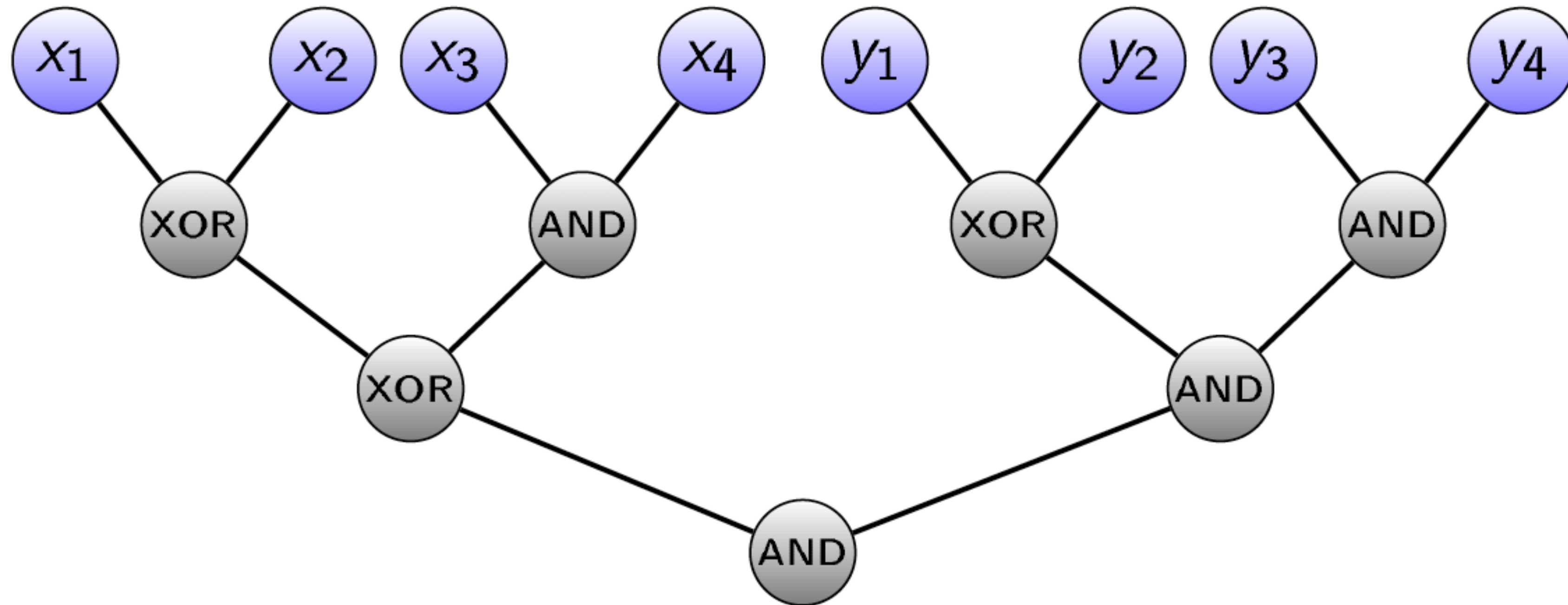
**Claim:** any polytime-computable function can be computed by a poly size boolean circuit over the {XOR, AND} bases.

**Proof:** that's how your computer does it.



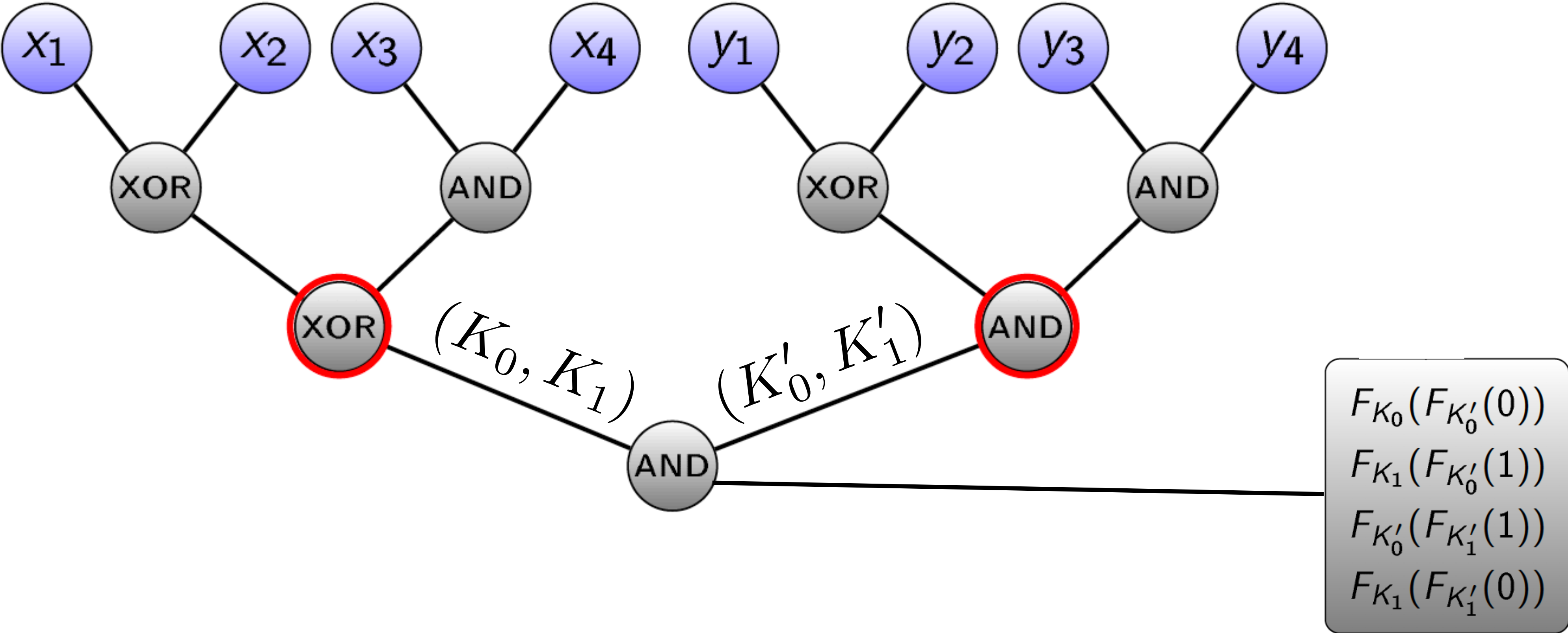
# Garbled Circuits

**Idea:** « encrypting » the gates such that they can only be evaluated given appropriate keys, and while hiding their exact behavior.

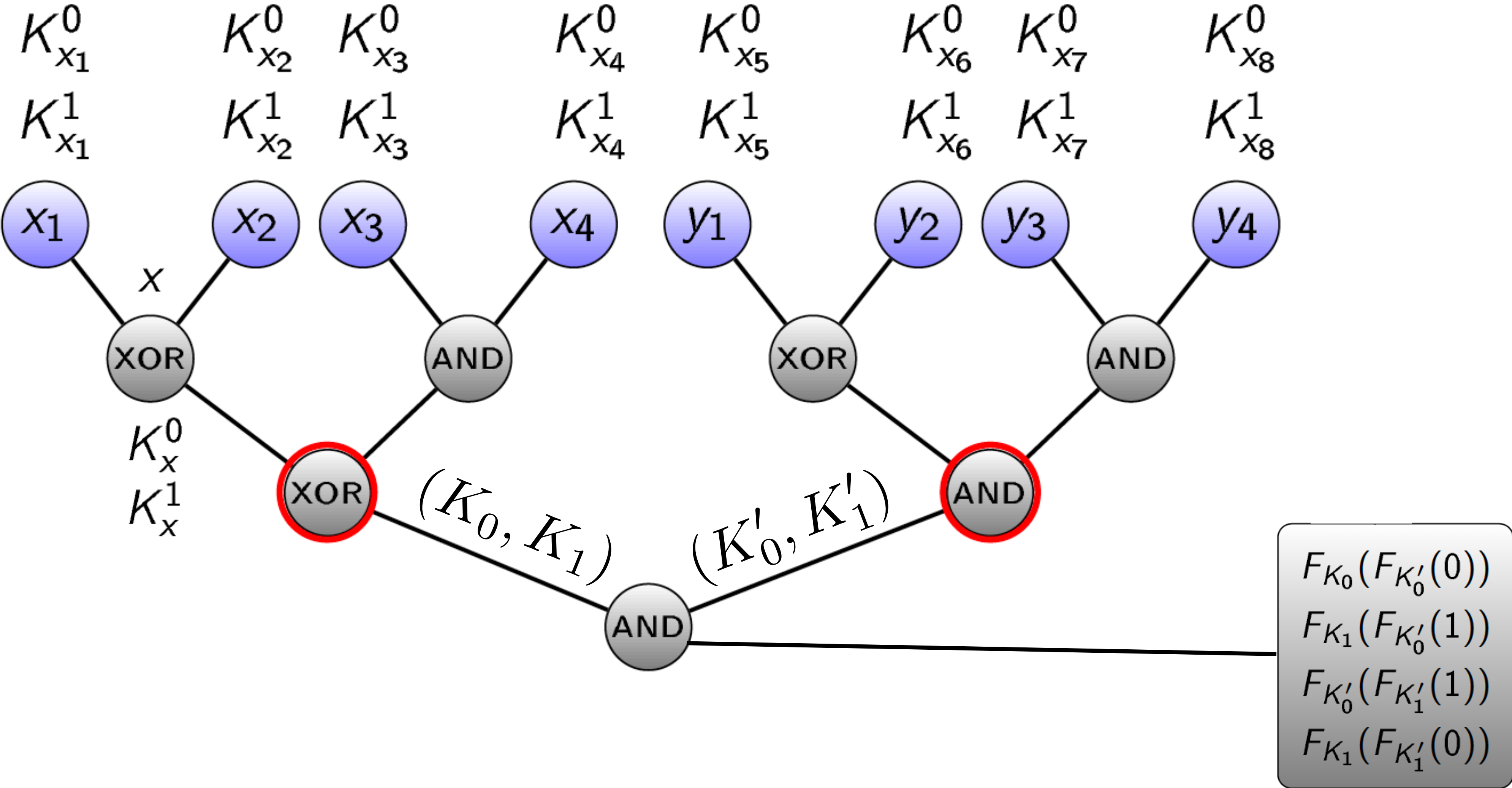


# Garbled Circuits

**Idea:** « encrypting » the gates such that they can only be evaluated given appropriate keys, and while hiding their exact behavior.

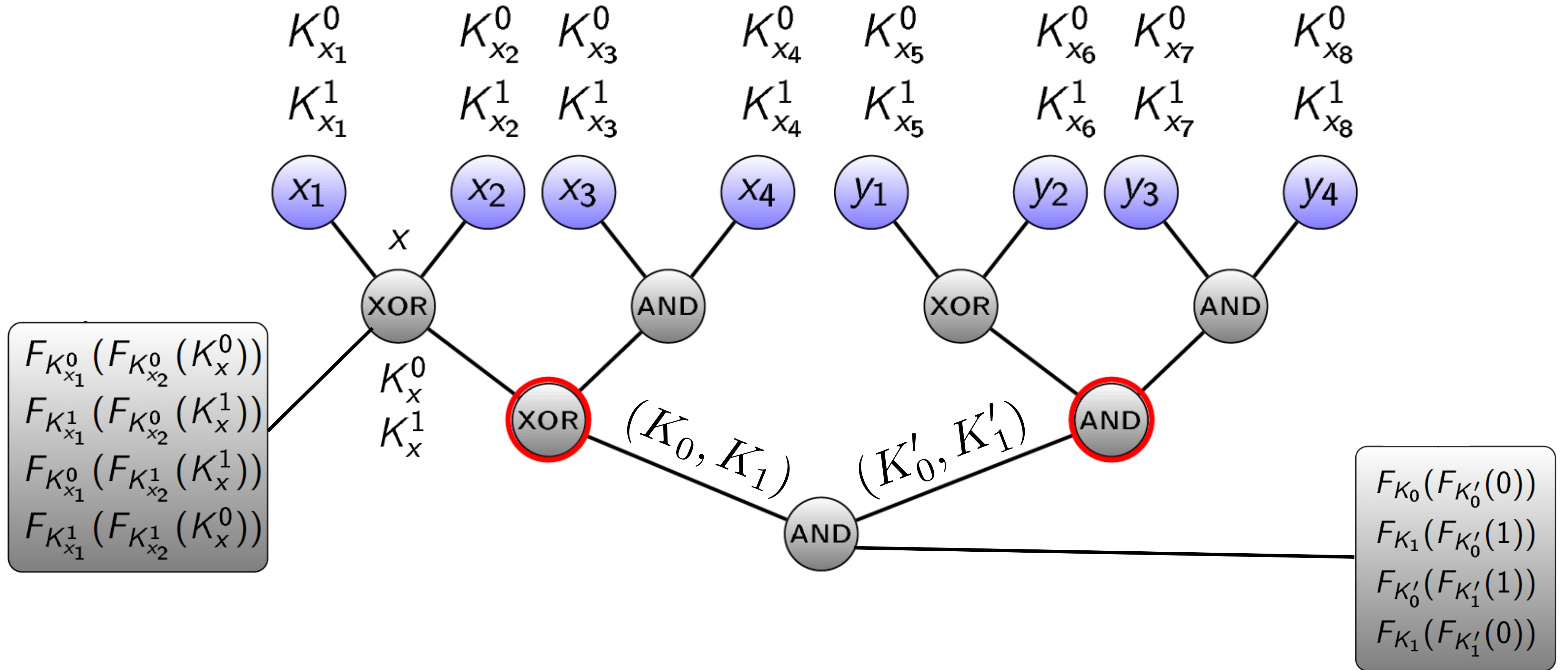


# Garbled Circuits

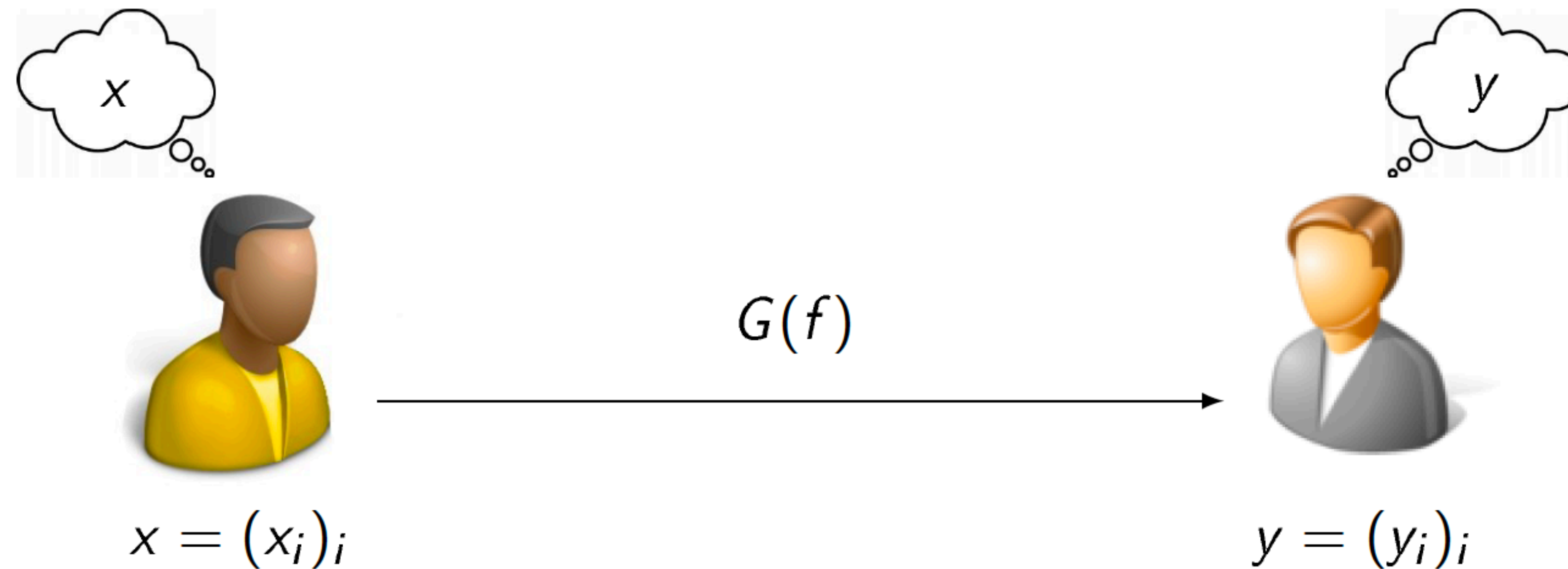




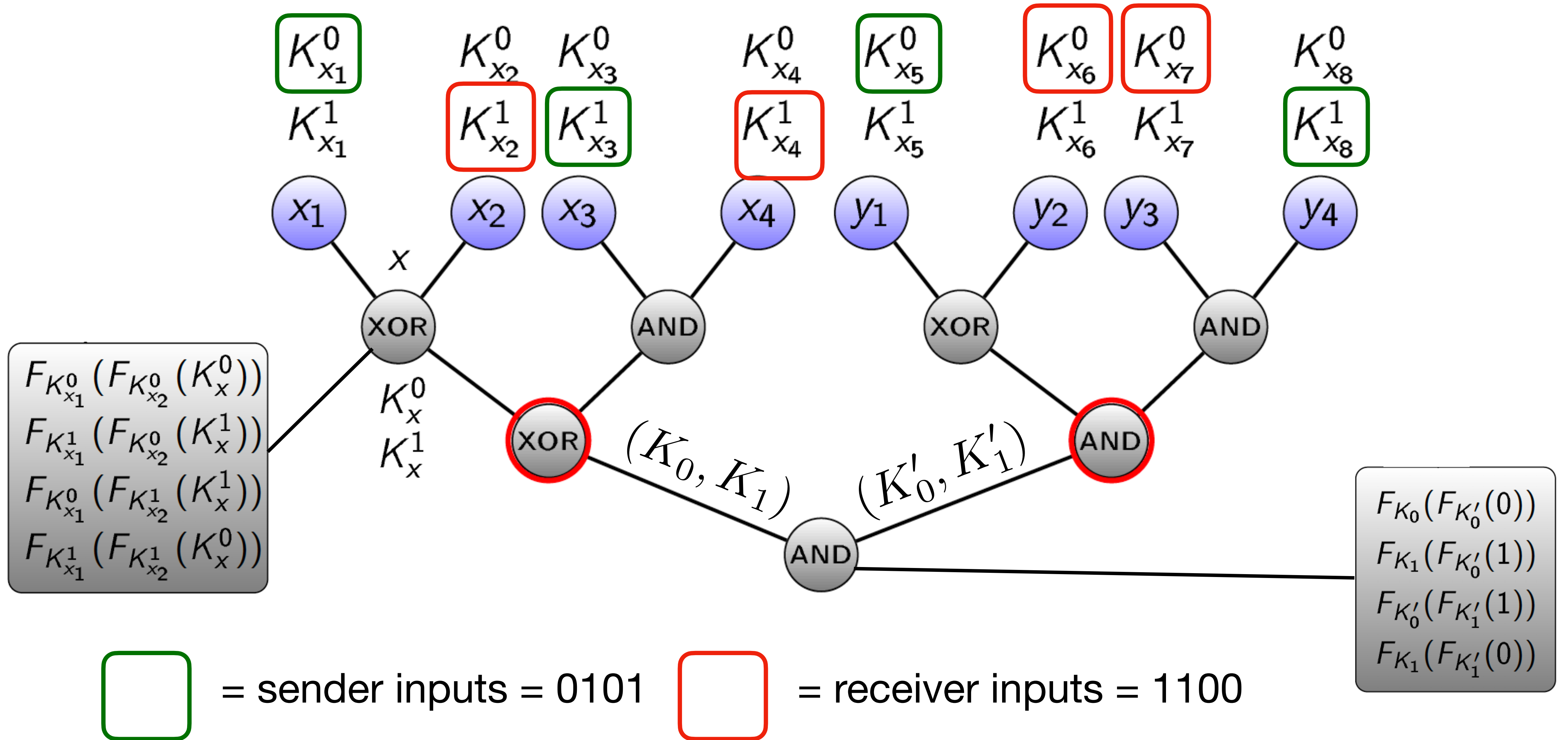
# Garbled Circuits



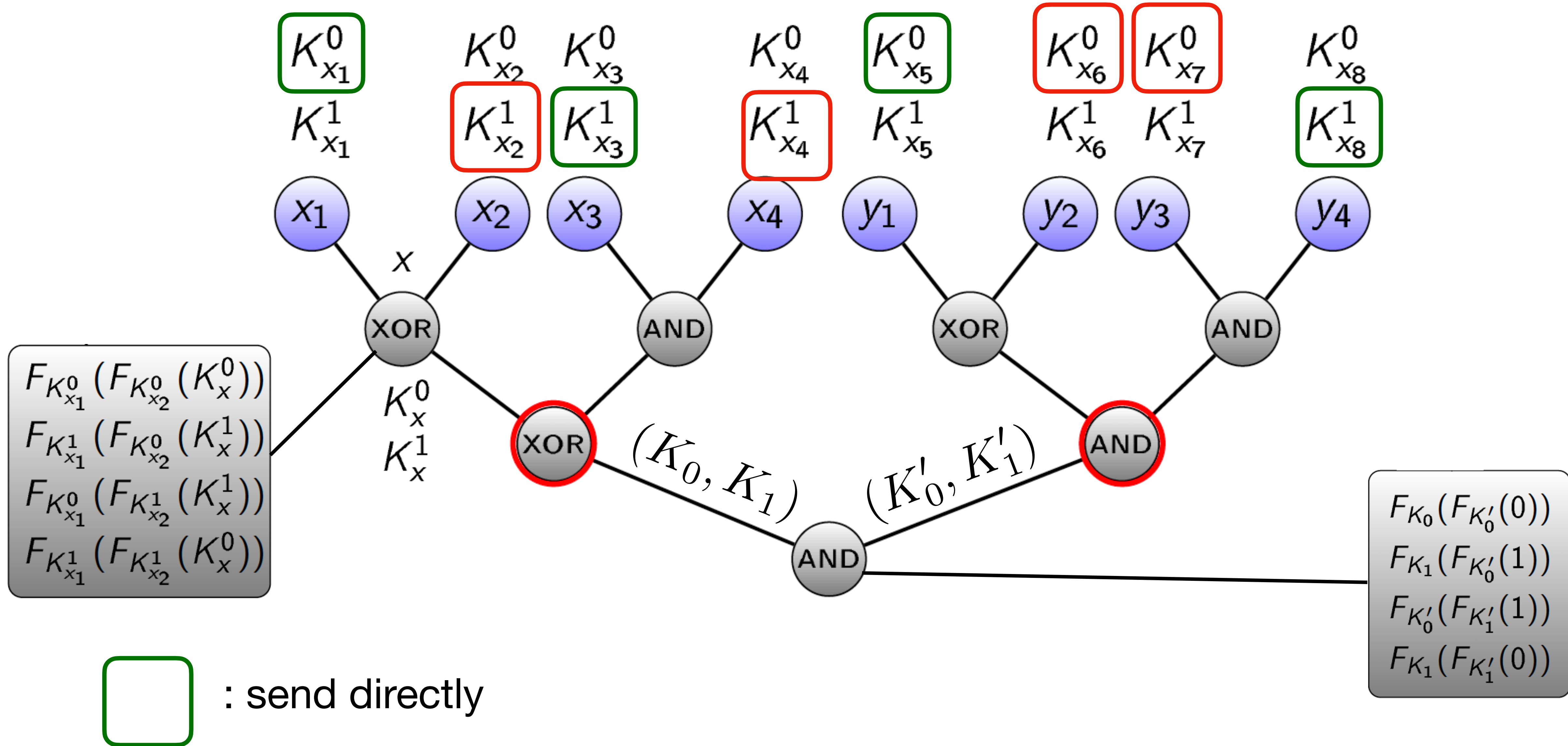
# Two-Party Secure Computation for All Functions



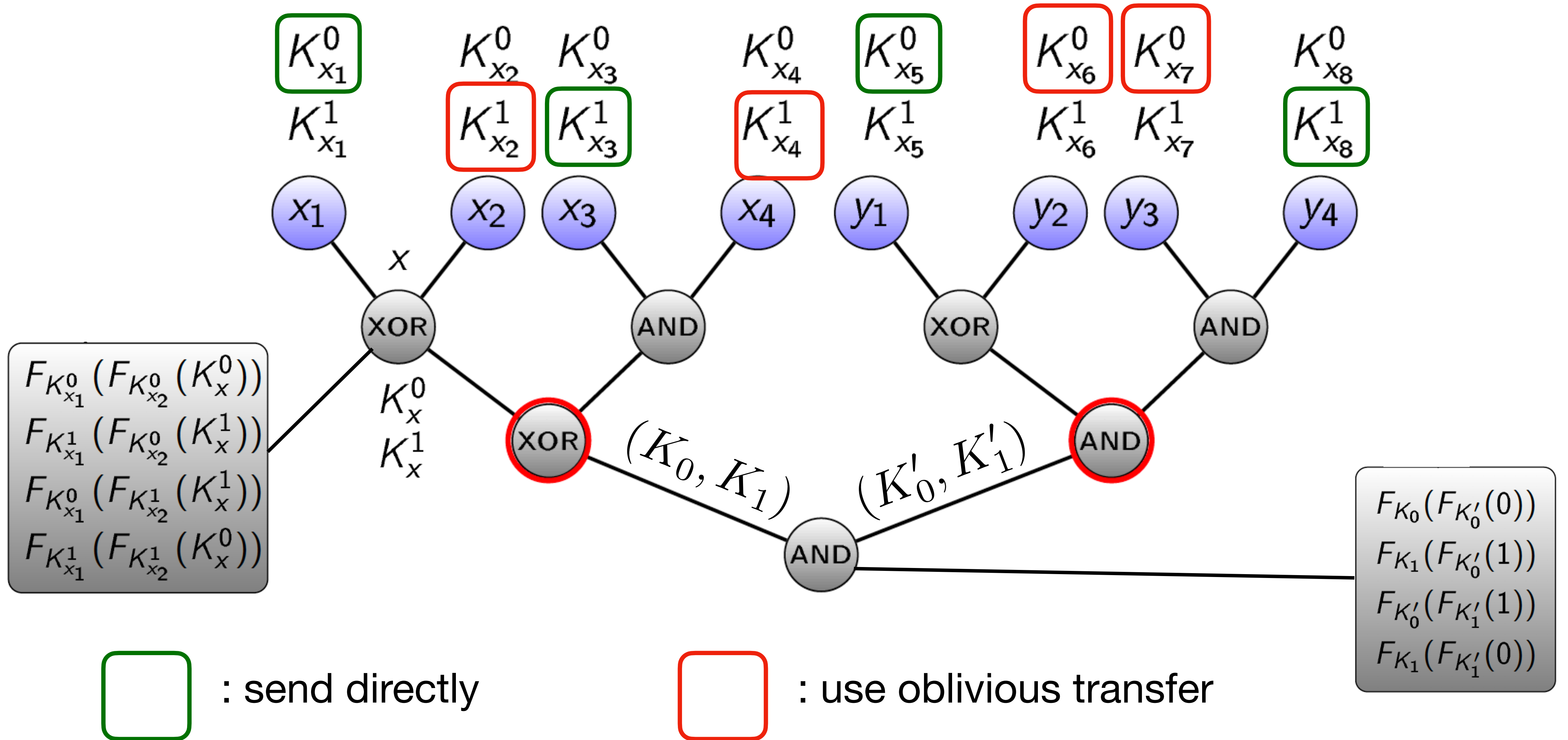
# Garbled Circuits



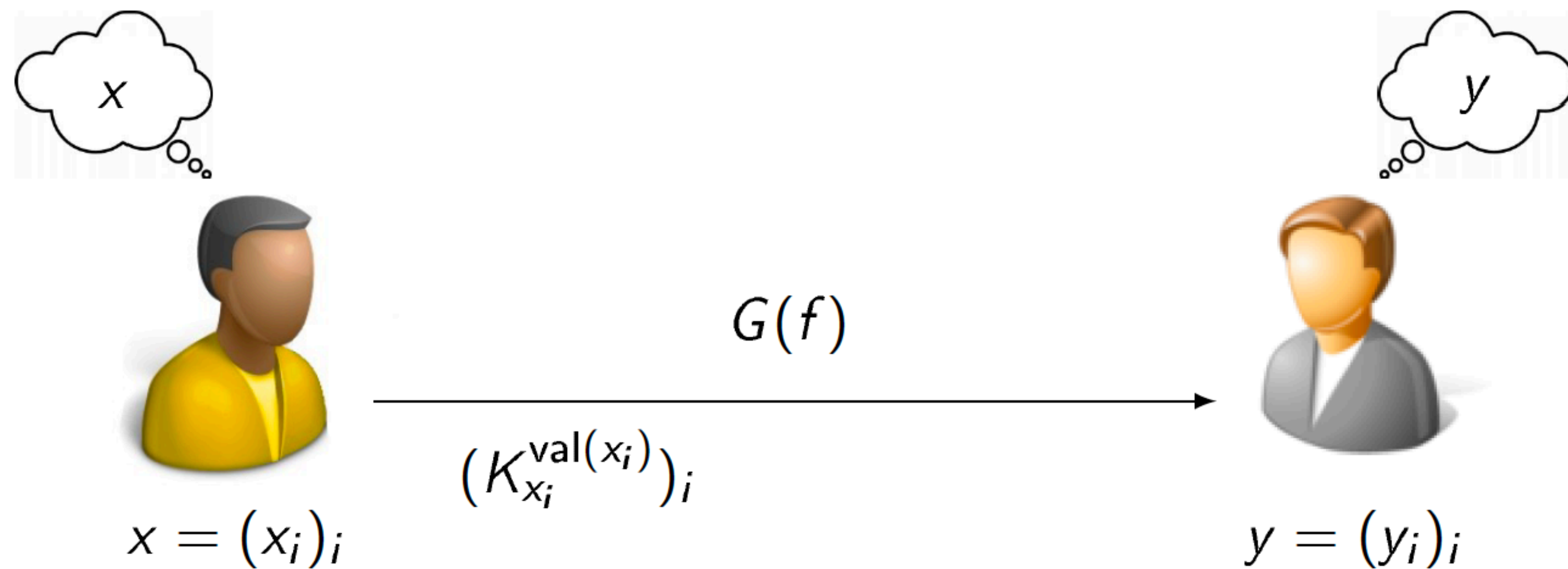
# Garbled Circuits



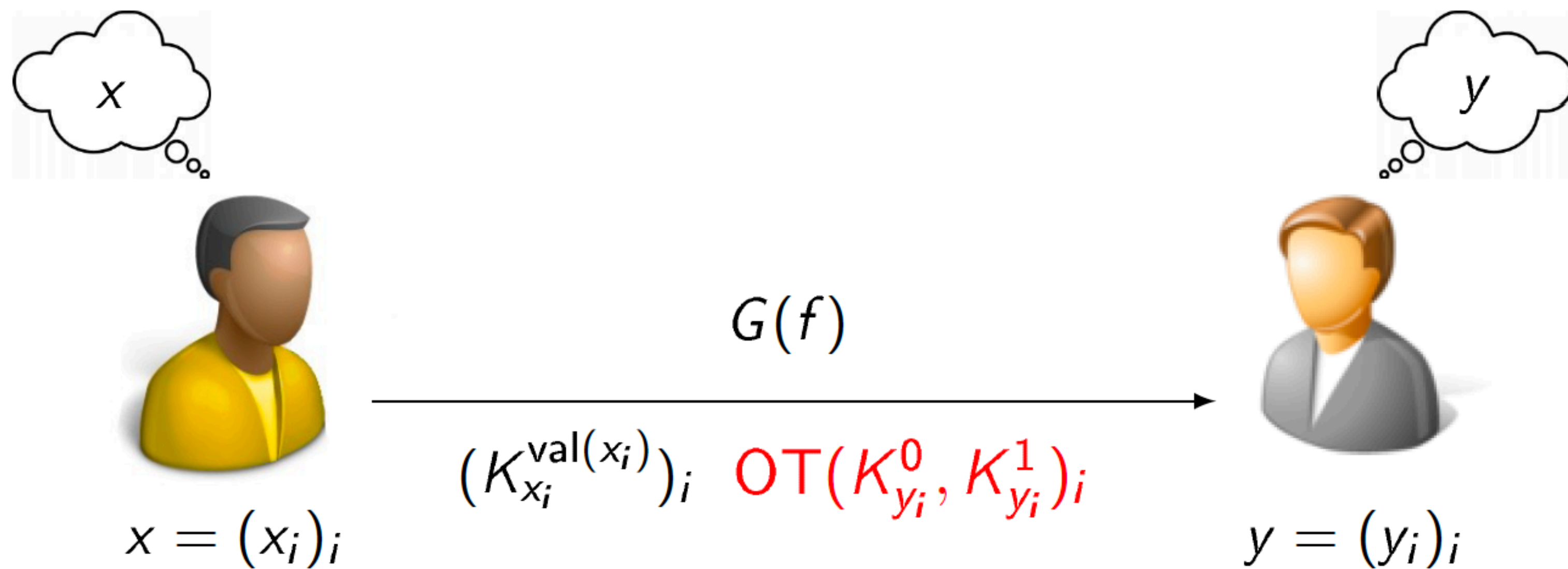
# Garbled Circuits



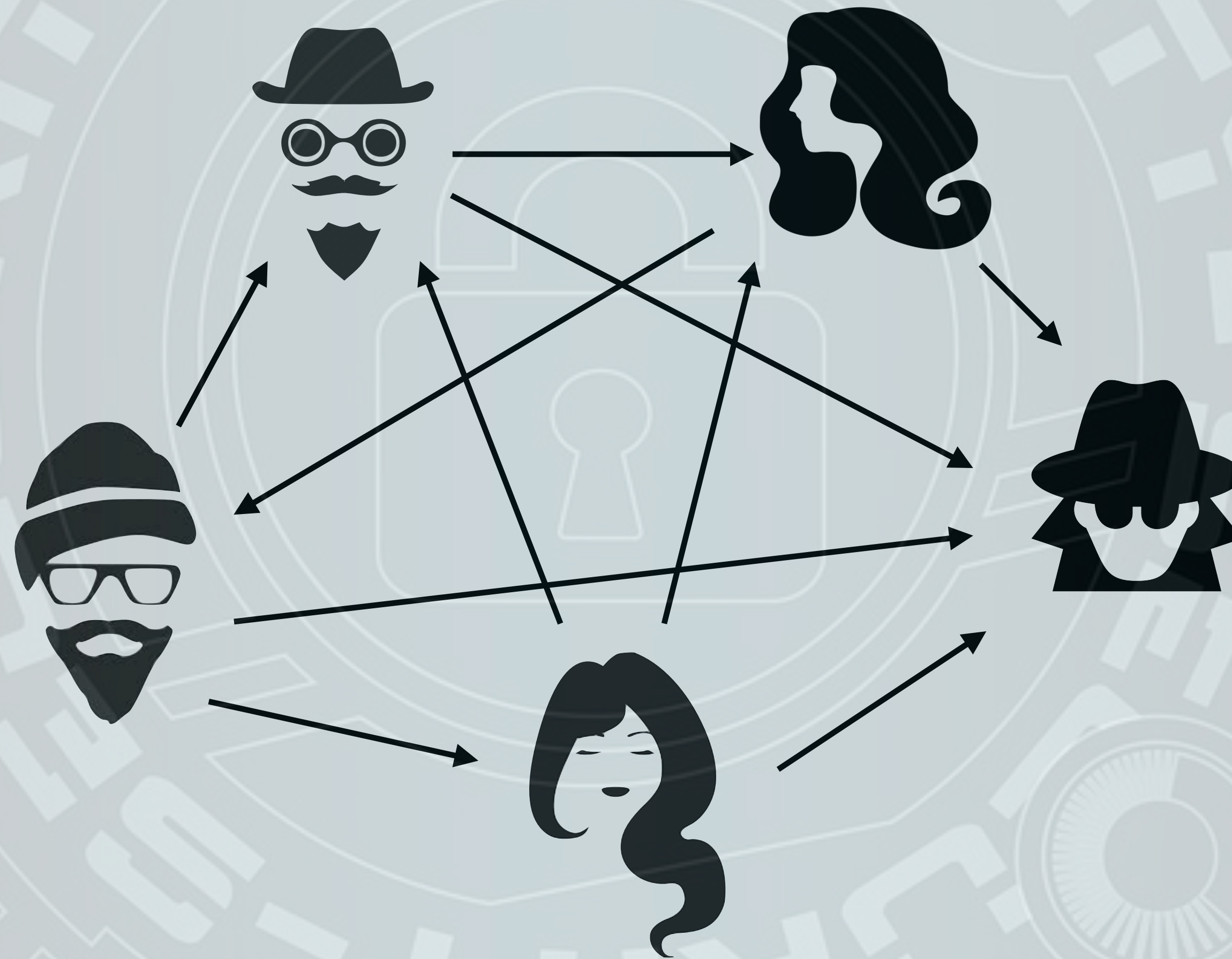
# Two-Party Secure Computation for All Functions



# Two-Party Secure Computation for All Functions



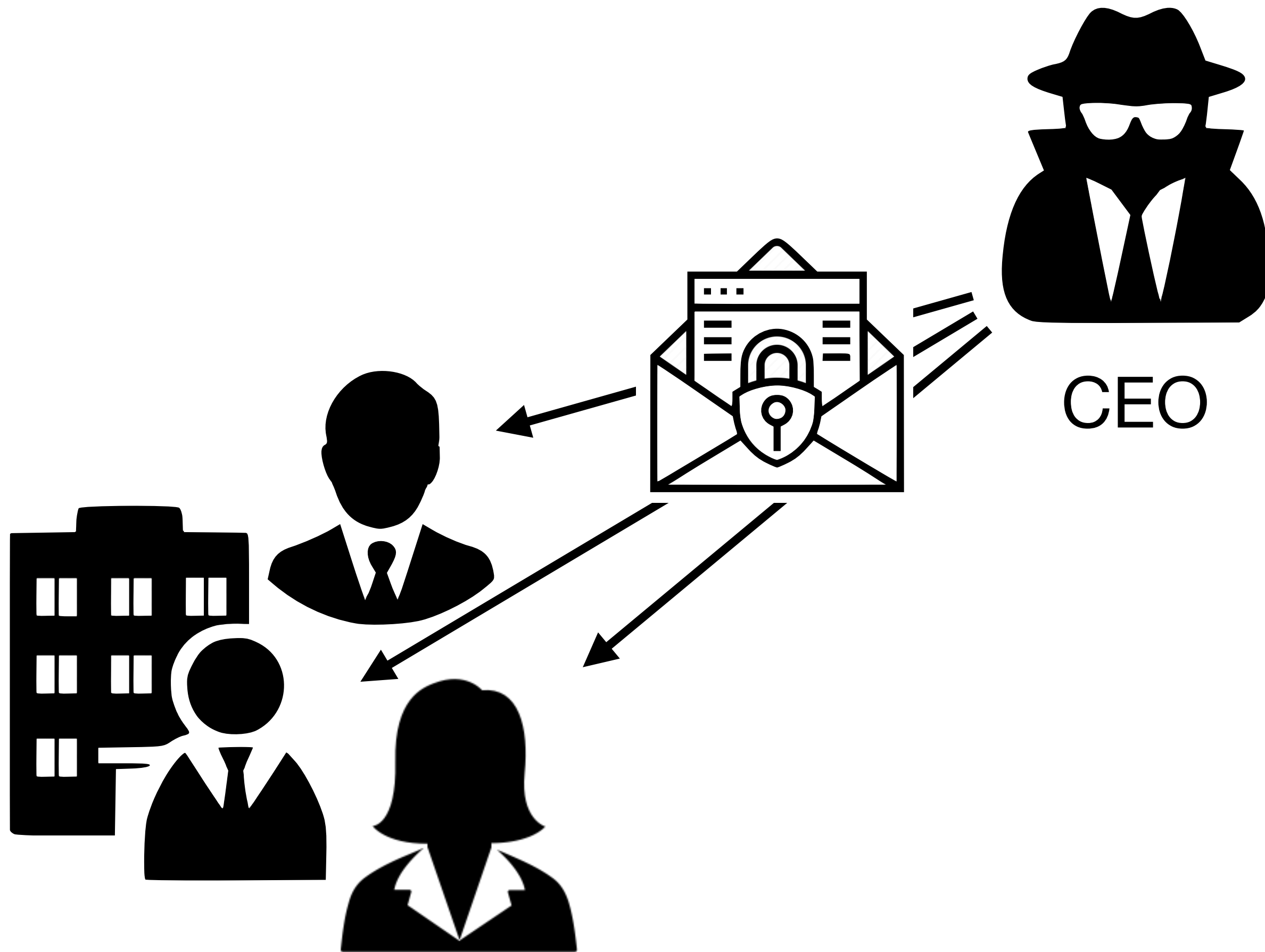
# Secure Computation with Many Players





# Secret Sharing

## The Problem - Concrete Version

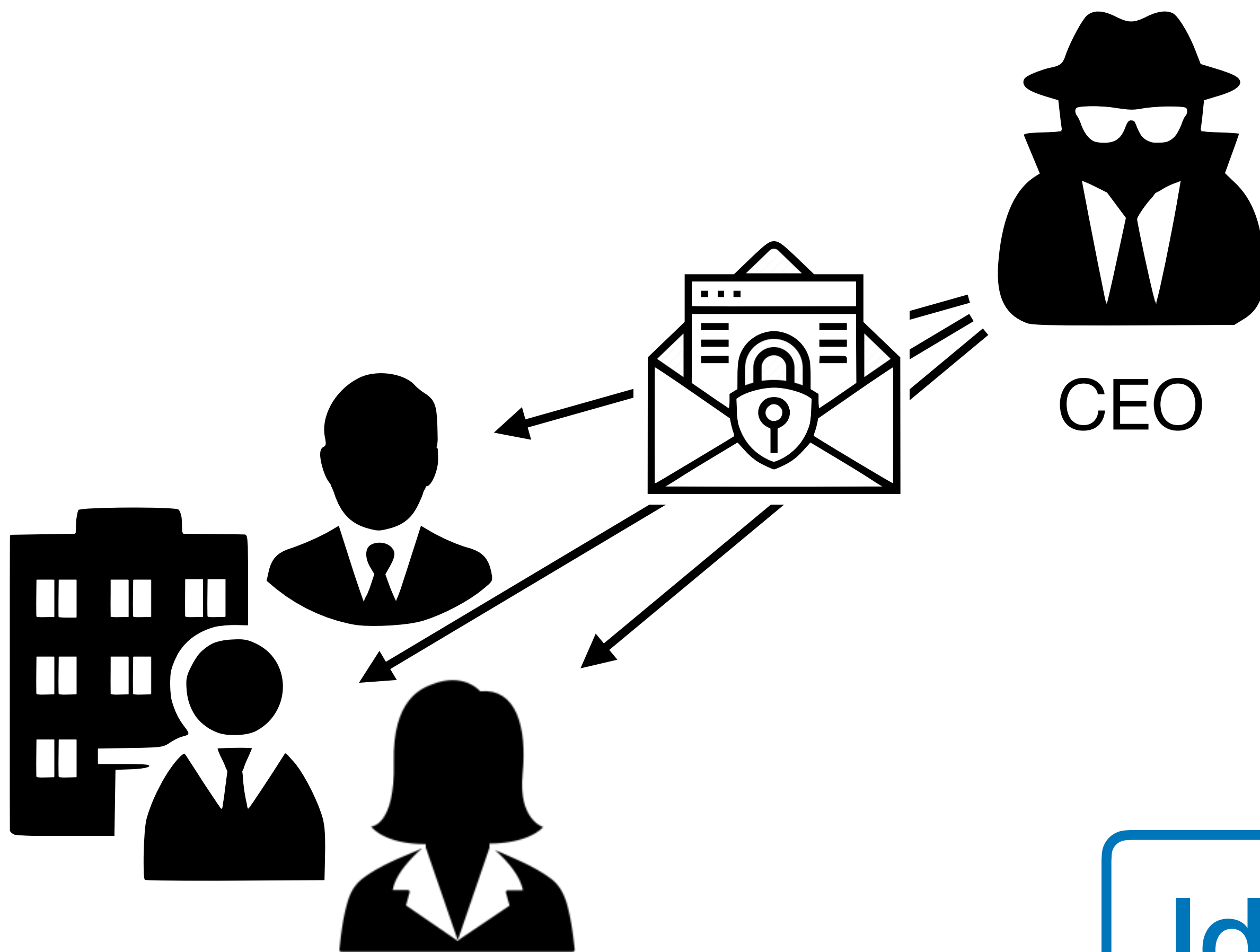


A company owns some very sensitive piece of information (e.g. a recipe, a business plan, etc). It has a board, composed of three members. The CEO wants to hand this information to the board, with the following guarantees:

- If two (or more) members of the board agree to reveal the secret information to someone else, then they can do it;
- However, no single board member should be able to leak *any* information about the secret, *even if s.he is completely malicious.*

# Secret Sharing

## The Problem - Concrete Version



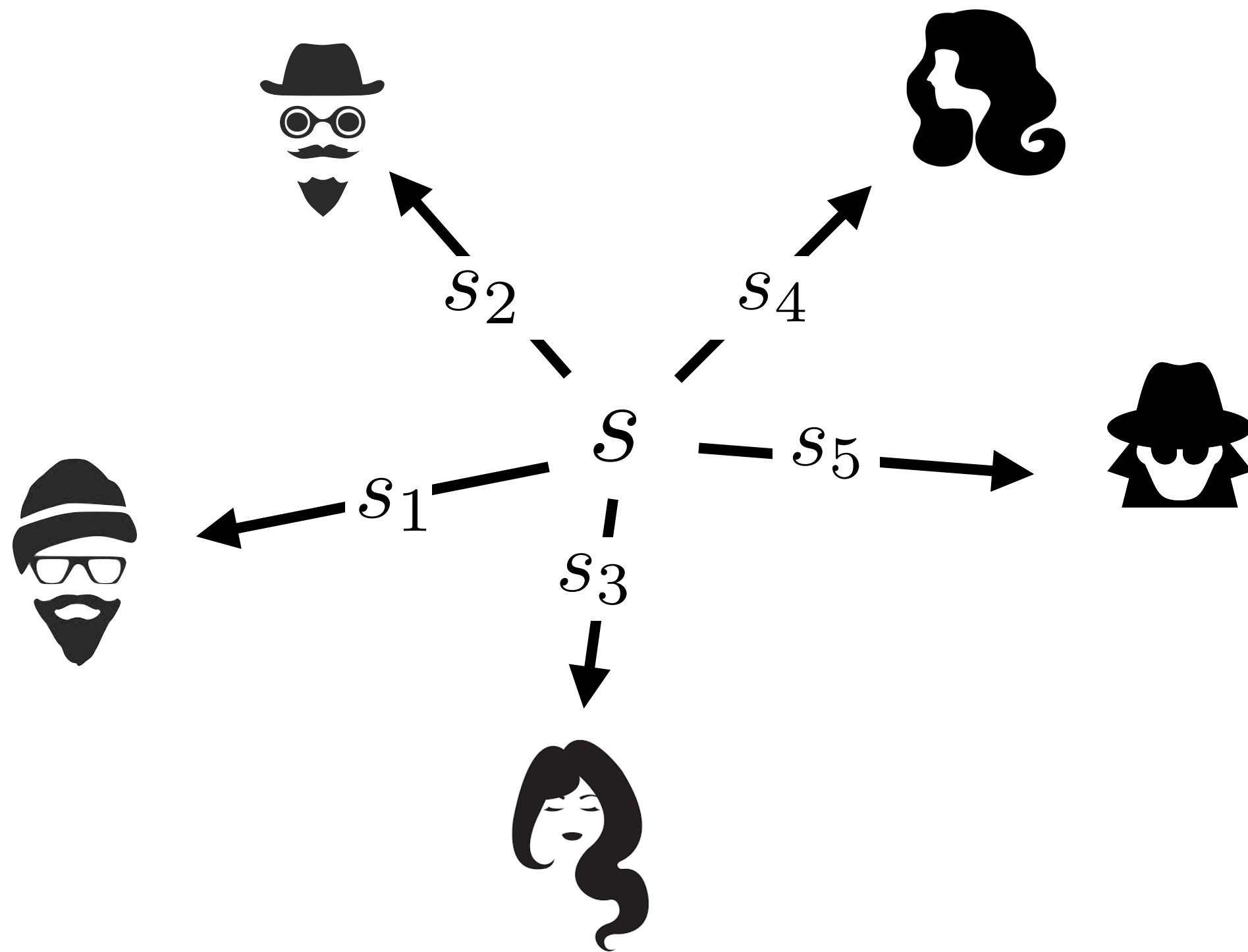
A company owns some very sensitive piece of information (e.g. a recipe, a business plan, etc). It has a board, composed of three members. The CEO wants to hand this information to the board, with the following guarantees:

- If two (or more) members of the board agree to reveal the secret information to someone else, then they can do it;
- However, no single board member should be able to leak *any* information about the secret, *even if s.he is completely malicious.*

Ideas?

# Secret Sharing

## The Problem - Abstract Version



- $(s_1, s_2, s_3, s_4, s_5) = \text{Share}(s)$
- $\text{Reconstruct}(\{s_i\}_{i \in Q}) = s$  iff  $|Q| \geq 3$
- $\{s_i\}_{i \in Q}$  leaks nothing about  $s$  if  $|Q| < 3$

### **(t,n)-secret sharing scheme**

We want to split a secret  $s$  between  $n$  parties such that:

- If at least  $t$  out of  $n$  parties collaborate, they can jointly *reconstruct* the secret  $s$ , but
- If strictly less than  $t$  parties collaborate, they learn *no information whatsoever* about  $s$

# (n,n)-Secret Sharing

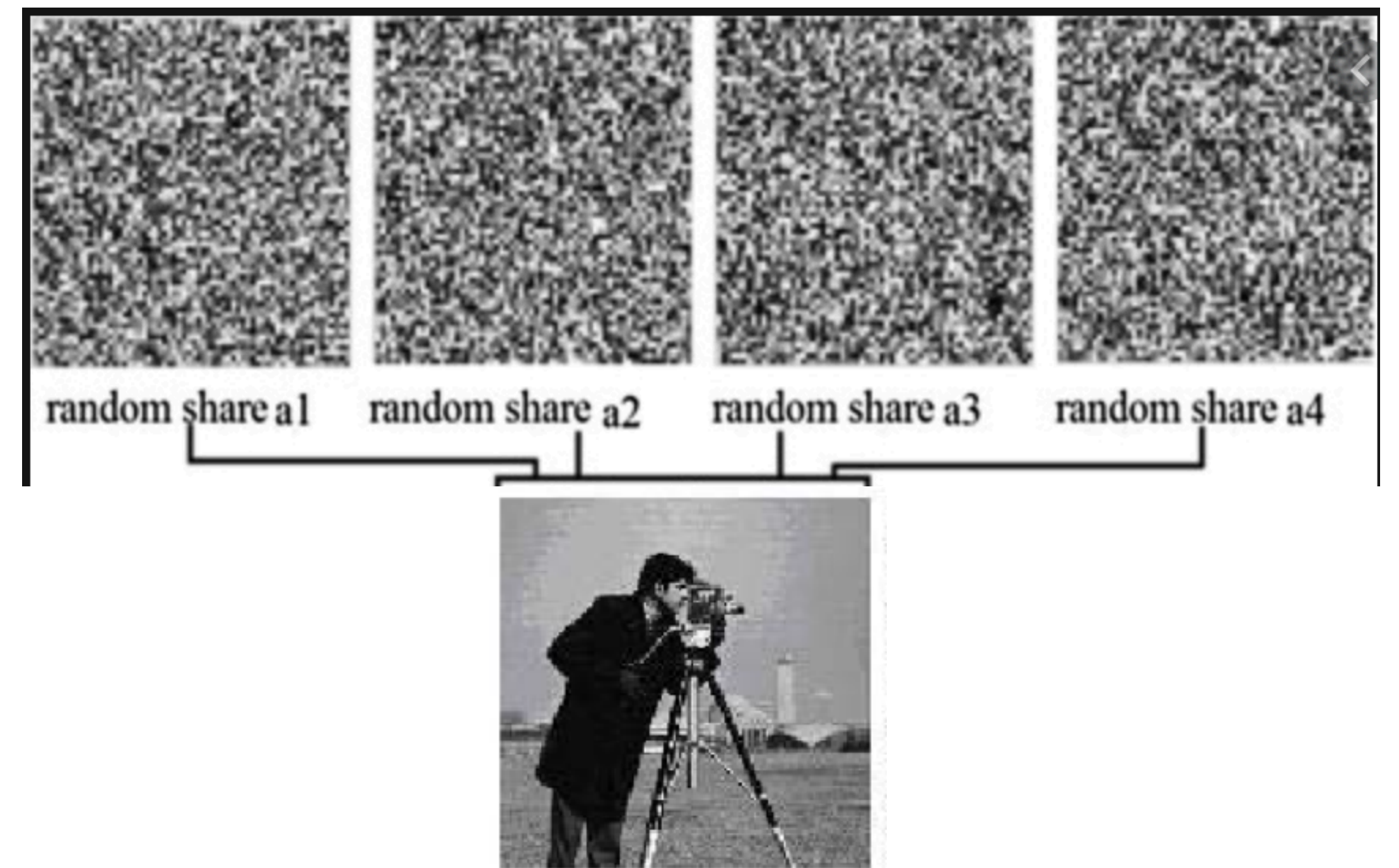
## A simple scheme:

Secret:  $s \in \{0, 1\}^\ell$

Share:  $\forall i \in [1, n - 1], s_i \leftarrow_r \{0, 1\}^\ell$   
 $s_n \leftarrow s_1 \oplus s_2 \oplus \dots \oplus s_{n-1} \oplus s$

Reconstruct:  $s = s_1 \oplus s_2 \oplus \dots \oplus s_n$

Important: convince yourself that  $\forall j, (s_i)_{i \neq j}$  leaks no information about  $s$



# (n,n)-Secret Sharing

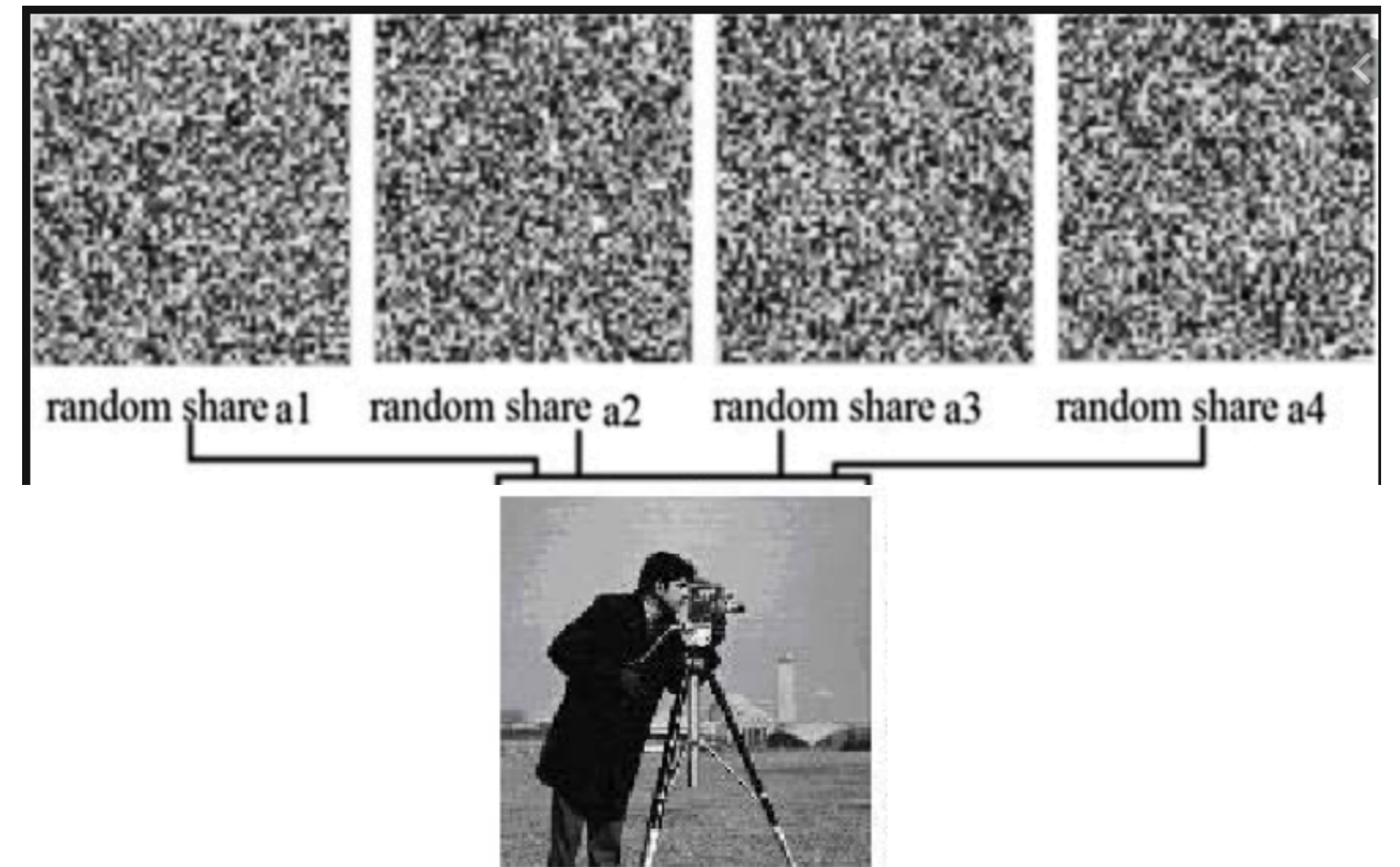
## A simple scheme:

Secret:  $s \in \{0, 1\}^\ell$

Share:  $\forall i \in [1, n - 1], s_i \leftarrow_r \{0, 1\}^\ell$   
 $s_n \leftarrow s_1 \oplus s_2 \oplus \dots \oplus s_{n-1} \oplus s$

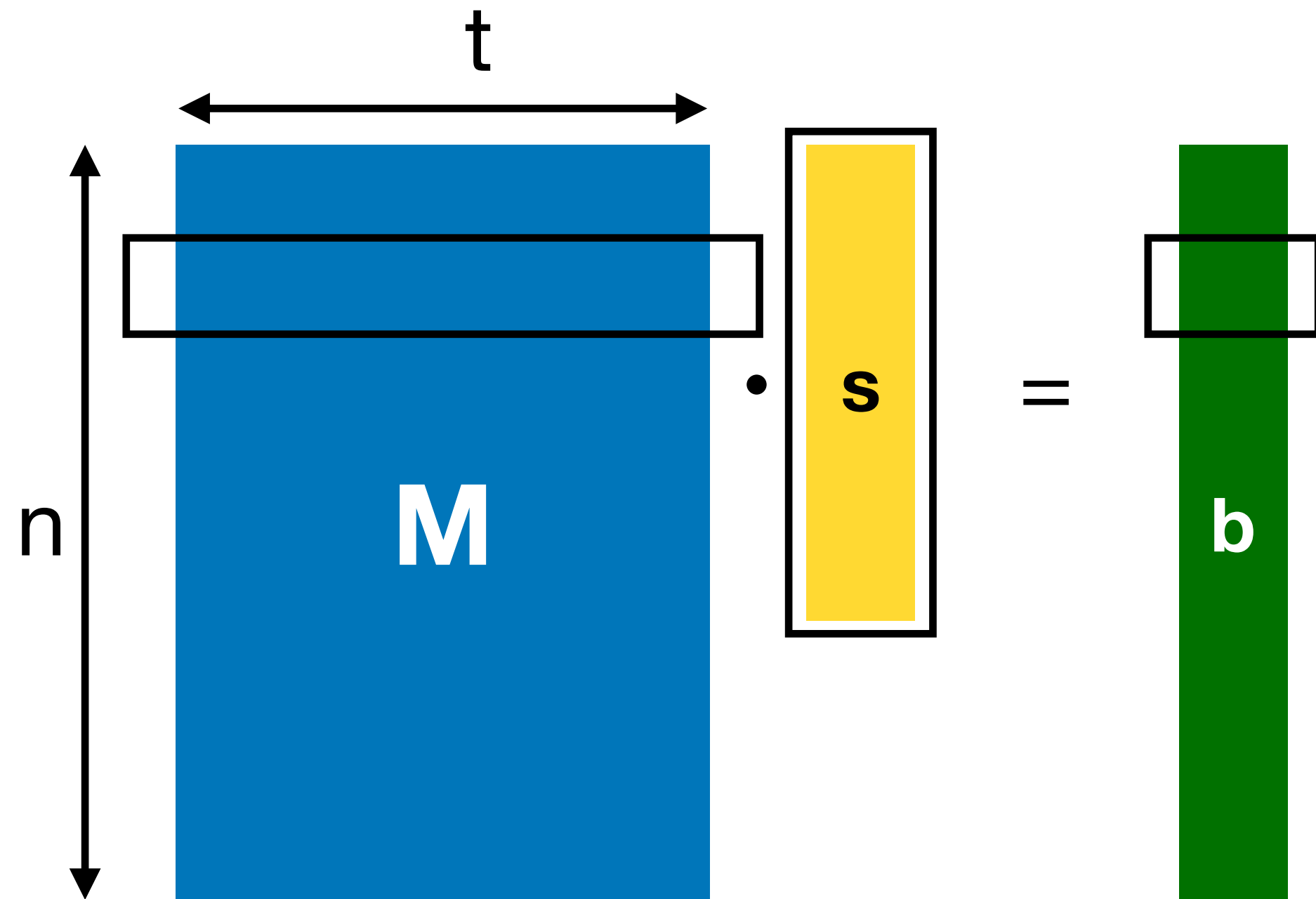
Reconstruct:  $s = s_1 \oplus s_2 \oplus \dots \oplus s_n$

Important: convince yourself that  $\forall j, (s_i)_{i \neq j}$  leaks no information about  $s$



What about (t,n)-sharing?

# Secret Sharing: Blakley's Scheme



Each party gets one equation

$< t$  parties gets an underdetermined system of equations  $\rightarrow$  the last entry of the vector  $s$  is undetermined.

$t$  parties have an invertible submatrix of  $M$  and can fully solve the system.

Secret:  $s \in \mathbb{F}_p$

Share:  $(s_1, \dots, s_{t-1}) \leftarrow_r \mathbb{F}_p^{t-1}$

$$\vec{s} = (s_1, \dots, s_{t-1}, s)$$

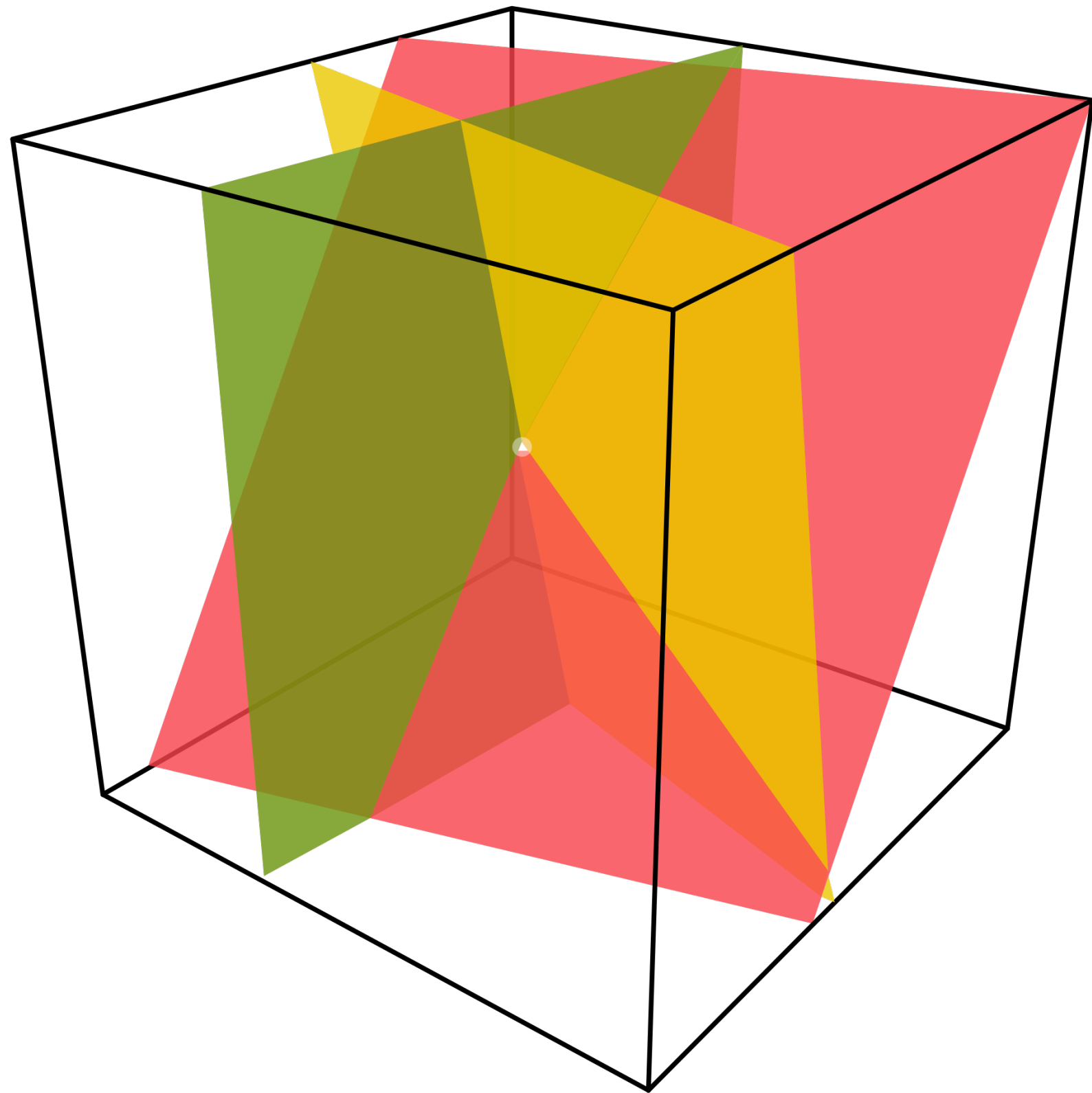
$$\vec{a}_i \leftarrow_r \mathbb{F}_p^t$$

$$b_i \leftarrow \langle \vec{a}_i, \vec{s} \rangle$$

← Share of player i.

Reconstruct: Gaussian elimination

# Secret Sharing: Blakley's Scheme



Equivalently: each party gets some hyperplane, and the secret is (a coordinate of) the only point at the intersection of all hyperplanes.

Secret:  $s \in \mathbb{F}_p$

Share:  $(s_1, \dots, s_{t-1}) \leftarrow_r \mathbb{F}_p^{t-1}$

$$\vec{s} = (s_1, \dots, s_{t-1}, s)$$

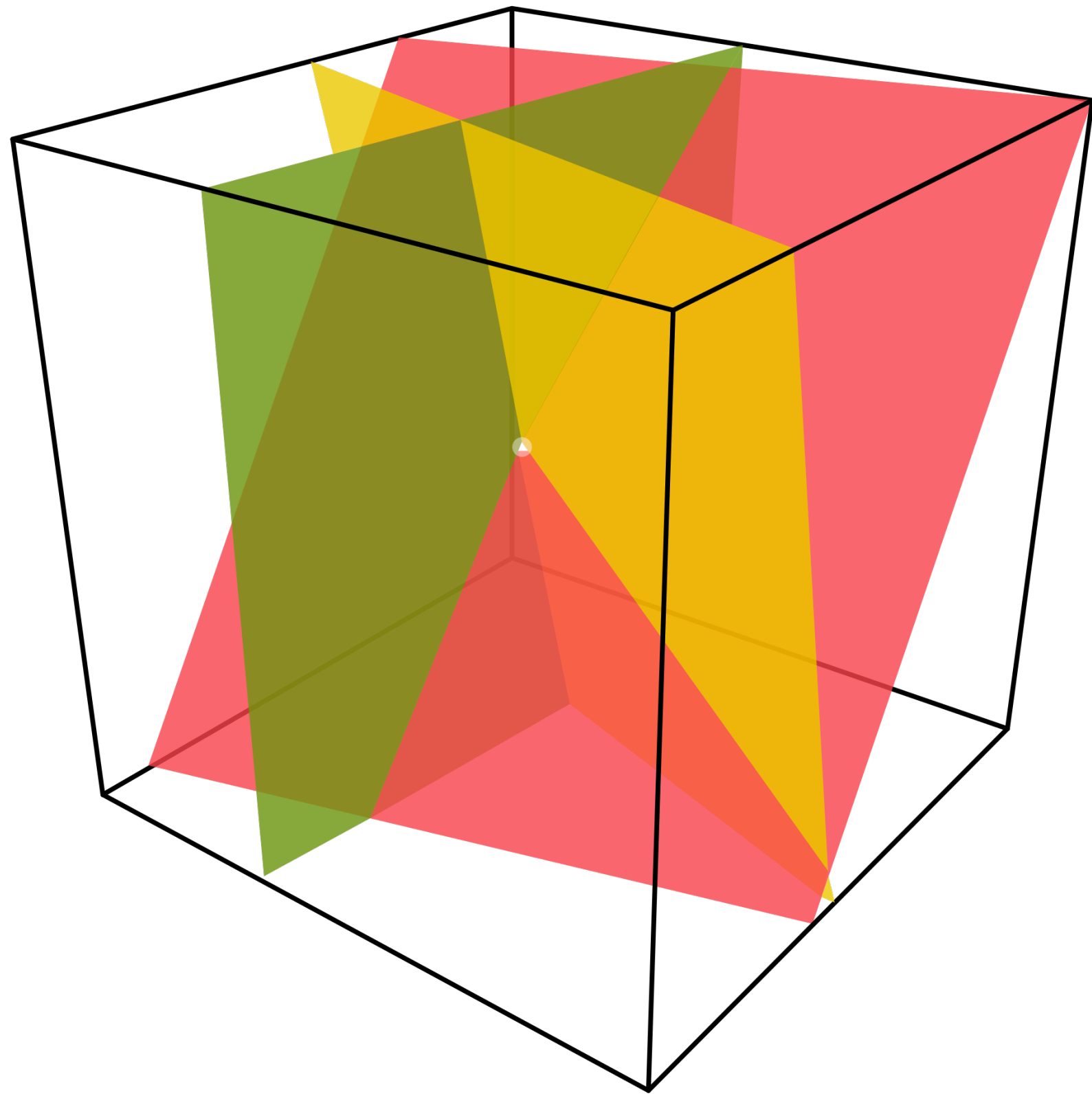
$$\vec{a}_i \leftarrow_r \mathbb{F}_p^t$$

$$b_i \leftarrow \langle \vec{a}_i, \vec{s} \rangle$$

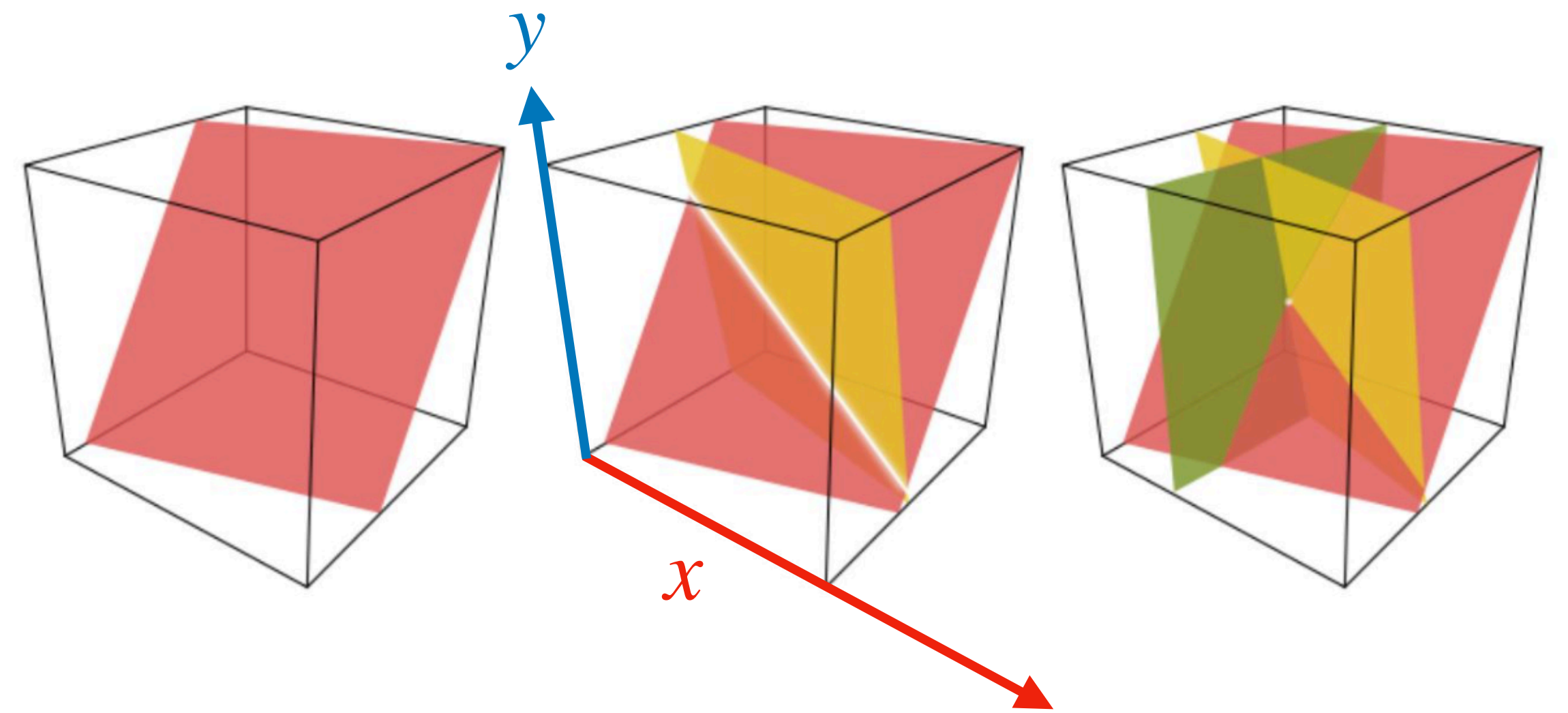
← Share of player i.

Reconstruct: Gaussian elimination

# Secret Sharing: Blakley's Scheme



Equivalently: each party gets some hyperplane, and the secret is (a coordinate of) the only point at the intersection of all hyperplanes.



The intersection of  $t$   $(t-1)$ -dimensional hyperplanes is always a single point, while the intersection of any smaller number leaves all possibilities totally identical for (say) the  $x$  coordinate.



# Secret Sharing: Blakley's Scheme

**The shares are large... Can we be more efficient?**

Also: it's not 100% clear how to construct the matrix  $M$ .

# Secret Sharing: Blakley's Scheme

Let's step back and think: we want that

- Given  $t$  points, you have the full information about something
- Given  $(t-1)$  points, there are still *many* remaining options.

Any idea what could possibly be a perfect fit?

# Secret Sharing: Blakley's Scheme

Let's step back and think: we want that

- Given  $t$  points, you have the full information about something
- Given  $(t-1)$  points, there are still *many* remaining options.

Any idea what could possibly be a perfect fit?

# Secret Sharing: Blakley's Scheme

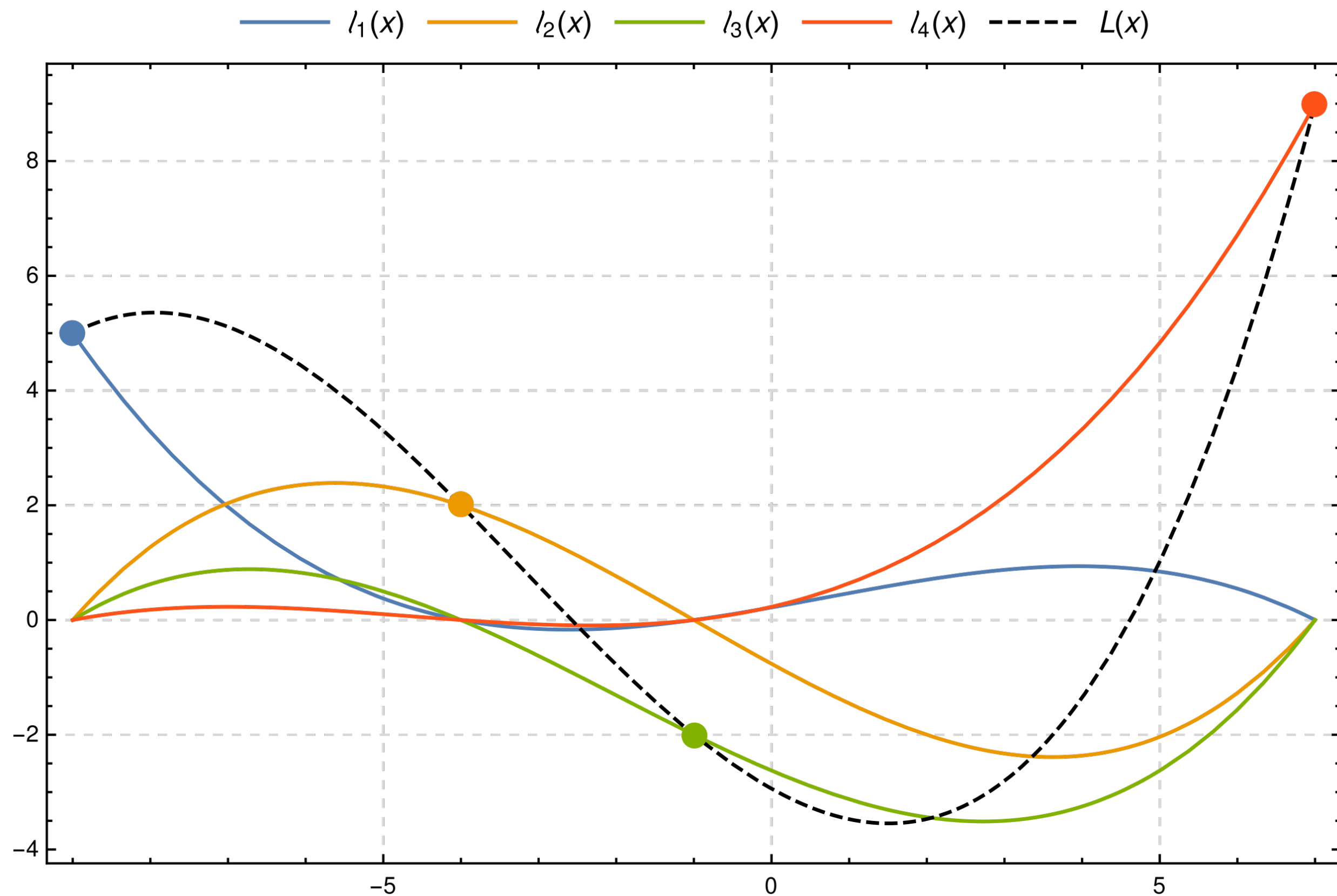
Let's step back and think: we want that

- Given  $t$  points, you have the full information about something
- Given  $(t-1)$  points, there are still *many* remaining options.

Any idea what could possibly be a perfect fit?

**Polynomial interpolation**

# Secret Sharing: Shamir's Scheme



Secret:  $s \in \mathbb{F}_p$

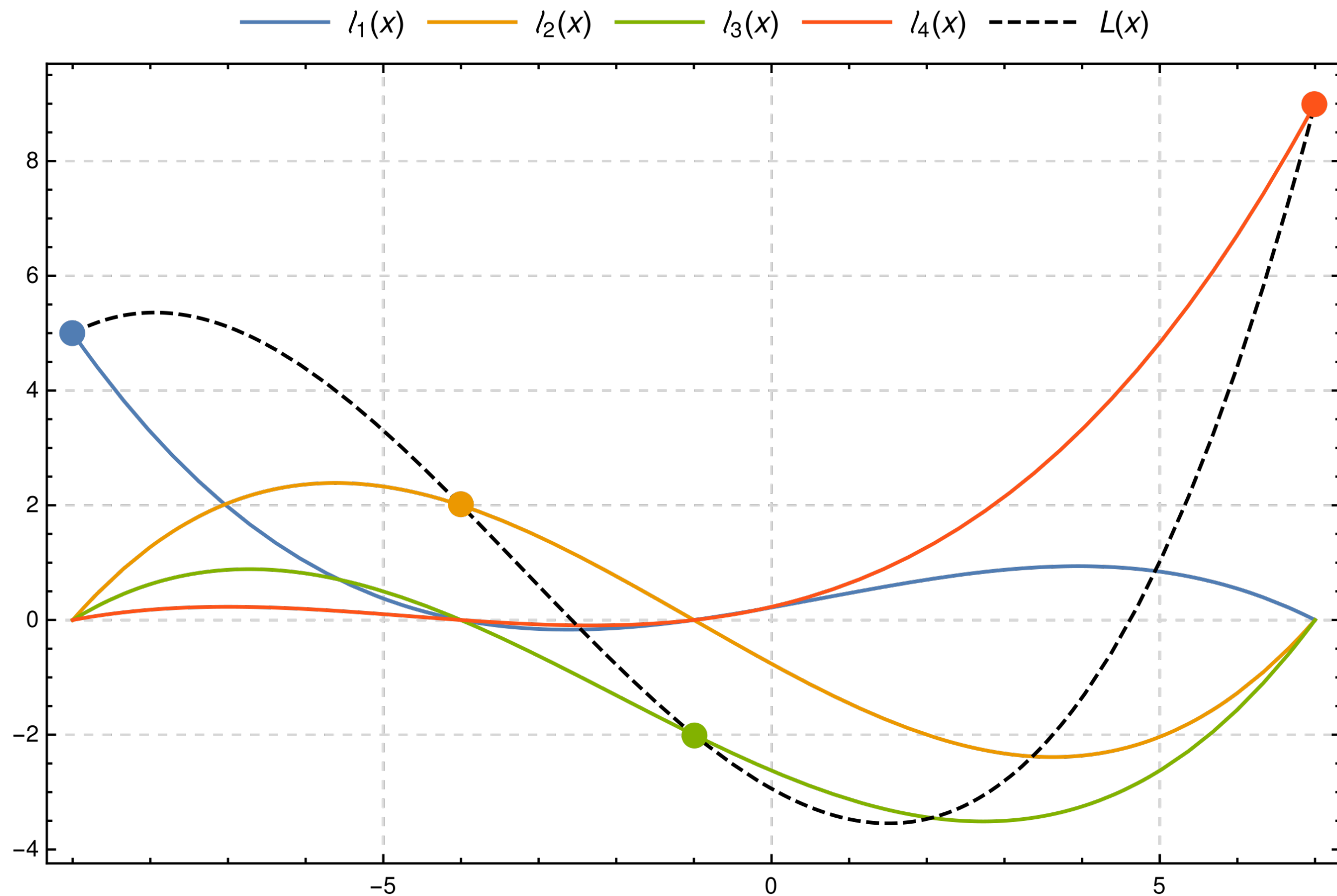
Share:  $(s_1, \dots, s_{t-1}) \leftarrow_r \mathbb{F}_p^{t-1}$

$$P_s(X) = s + \sum_{i=1}^{t-1} s_i \cdot X^i$$

$$P(1), P(2), \dots, P(n)$$

Reconstruct: Lagrange interpolation!

# Secret Sharing: Shamir's Scheme



Secret:  $s \in \mathbb{F}_p$

Share:  $(s_1, \dots, s_{t-1}) \leftarrow_r \mathbb{F}_p^{t-1}$

$$P_s(X) = s + \sum_{i=1}^{t-1} s_i \cdot X^i$$

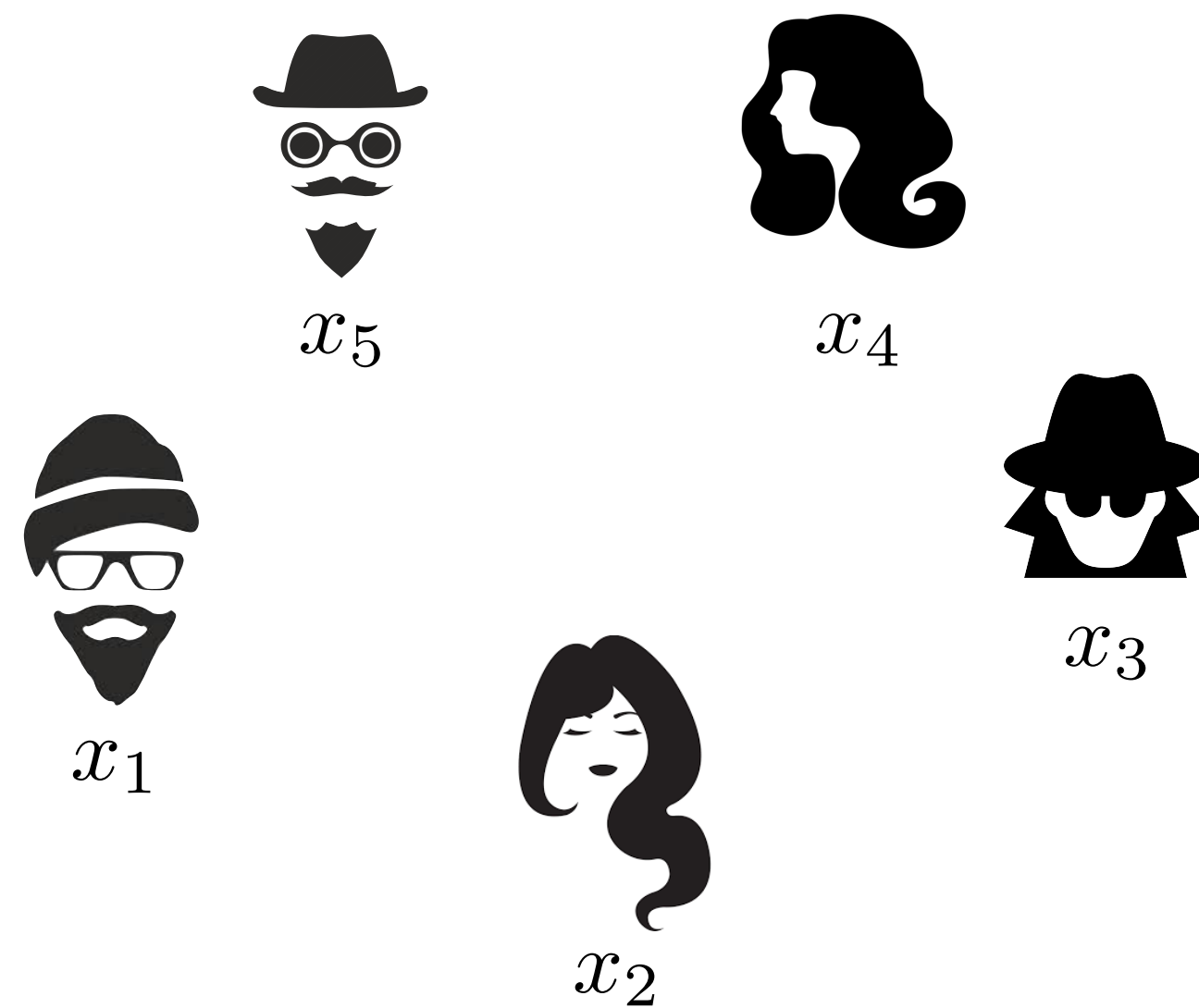
$$P(1), P(2), \dots, P(n)$$

Reconstruct: Lagrange interpolation!

**A share is a single field element**

# The GMW Protocol

## Reminder: the Model

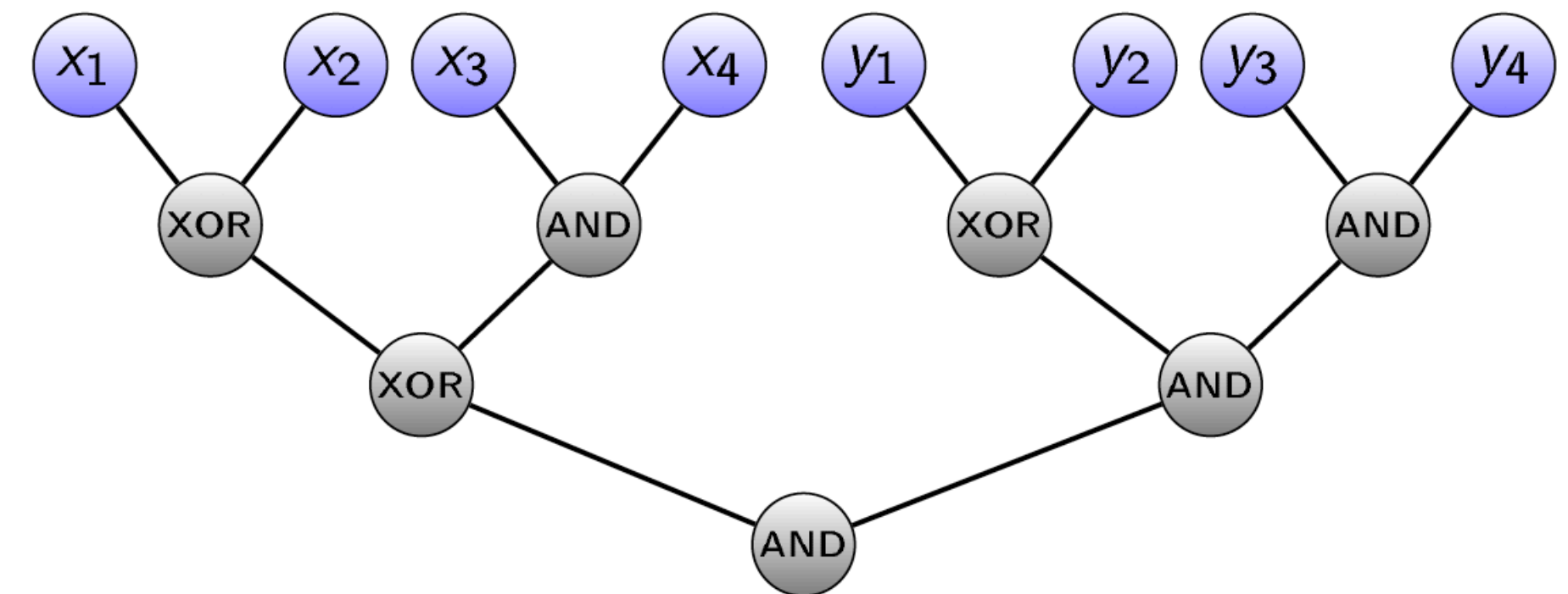


### Goal

- Public function  $f$
- All players want to get  $f(x_1, x_2, x_3, x_4, x_5)$
- No player should learn anything more

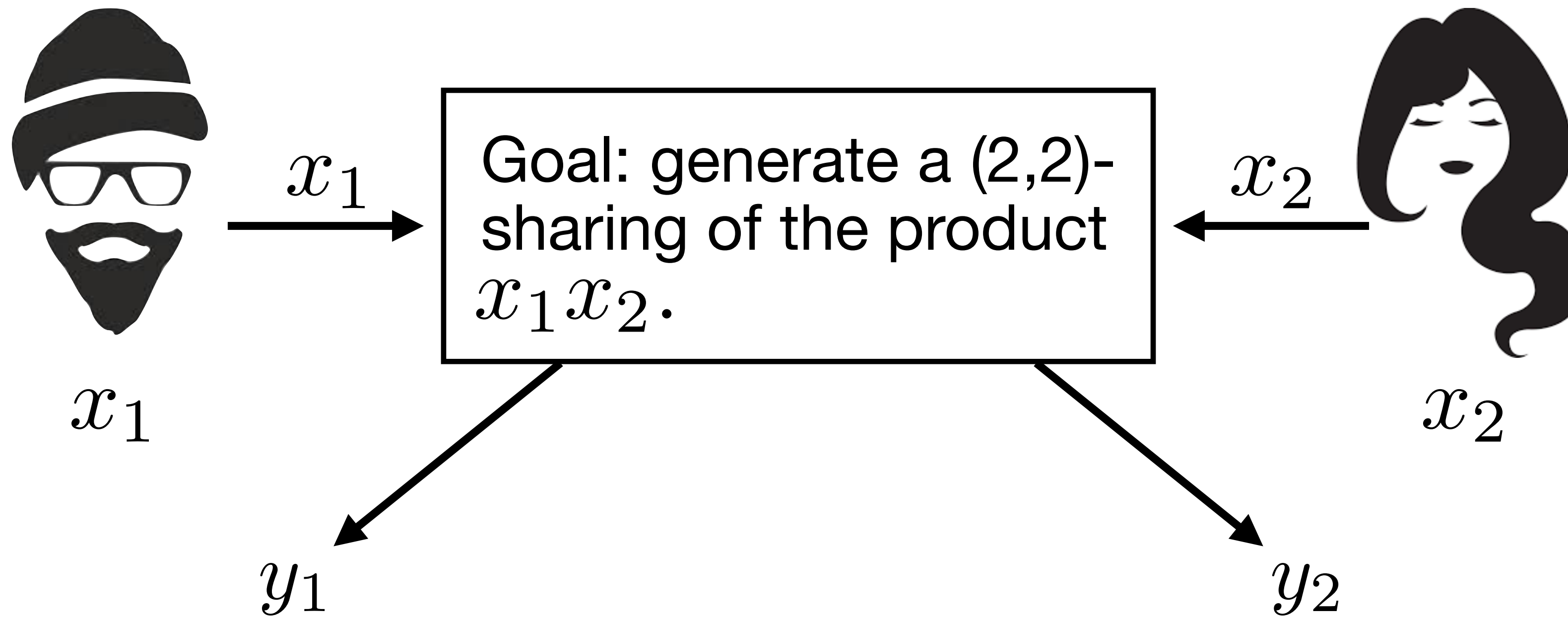
## Previously...

- We solved the 2-party case with garbled circuits
- Our function  $f$  was seen as a boolean circuit:



We will now see how to handle the general case of secure computation between  $n$  players, step by step. This protocol will rely crucially on the two ingredients we have seen: a computational ingredient (oblivious transfer) and an information theoretic ingredient (secret sharing).

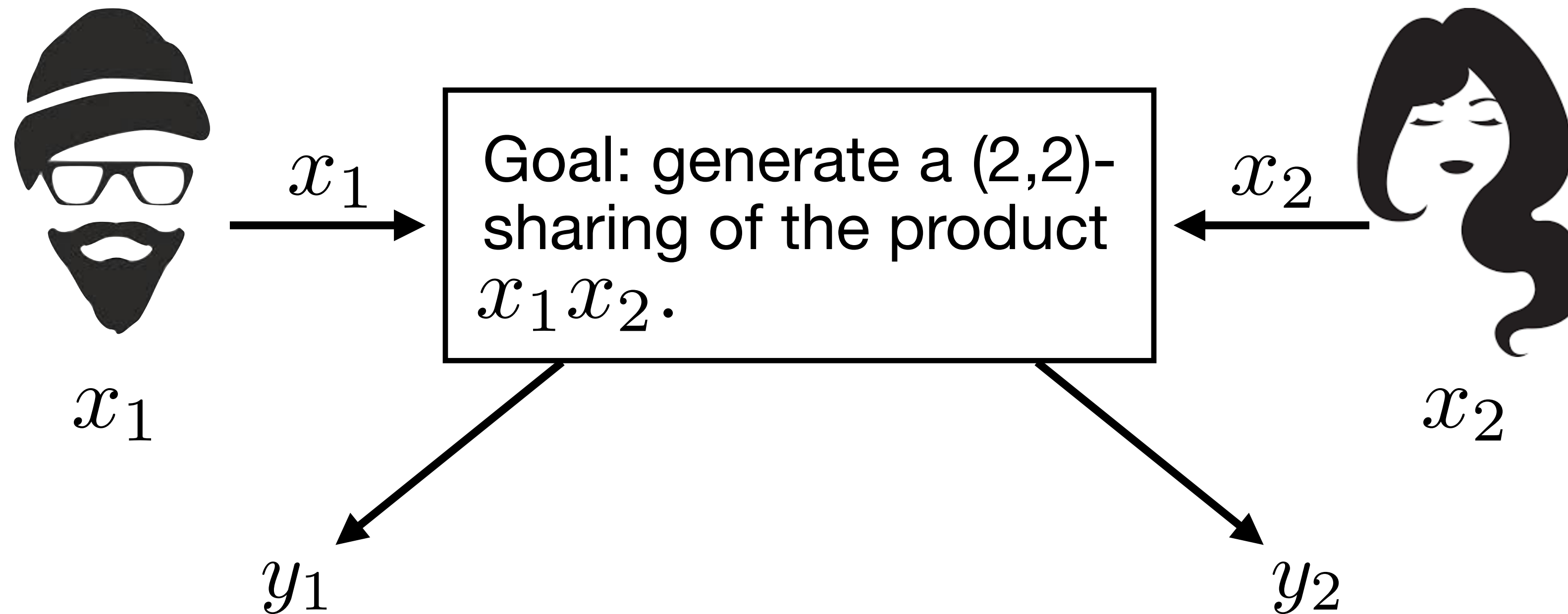
# Warm-up I: 2-Party Product Sharing



$(y_1, y_2)$  random conditioned on  $y_1 \oplus y_2 = x_1 x_2$



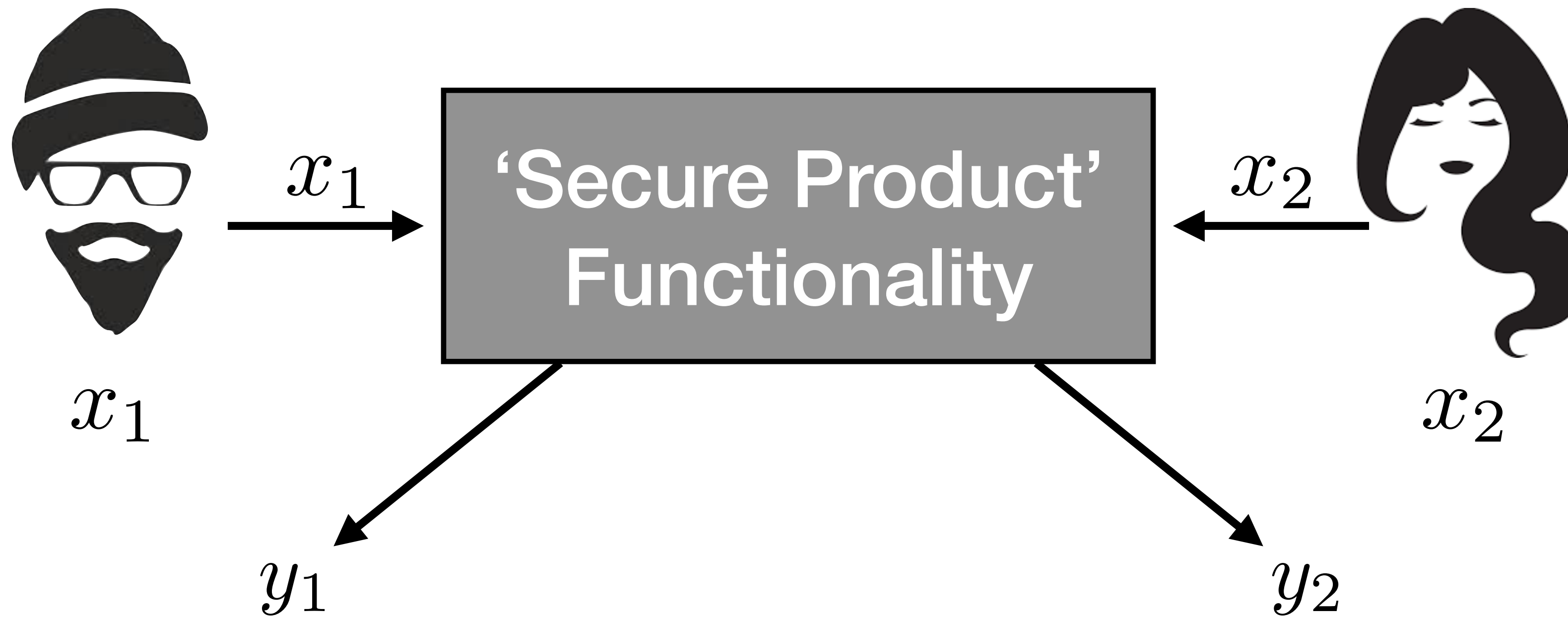
# Warm-up I: 2-Party Product Sharing



$(y_1, y_2)$  random conditioned on  $y_1 \oplus y_2 = x_1 x_2$

**Exercise: build this protocol**

# Warm-up I: 2-Party Product Sharing



$(y_1, y_2)$  random conditioned on  $y_1 \oplus y_2 = x_1 x_2$

**Exercise: build this protocol**

# Step-by Step Solution



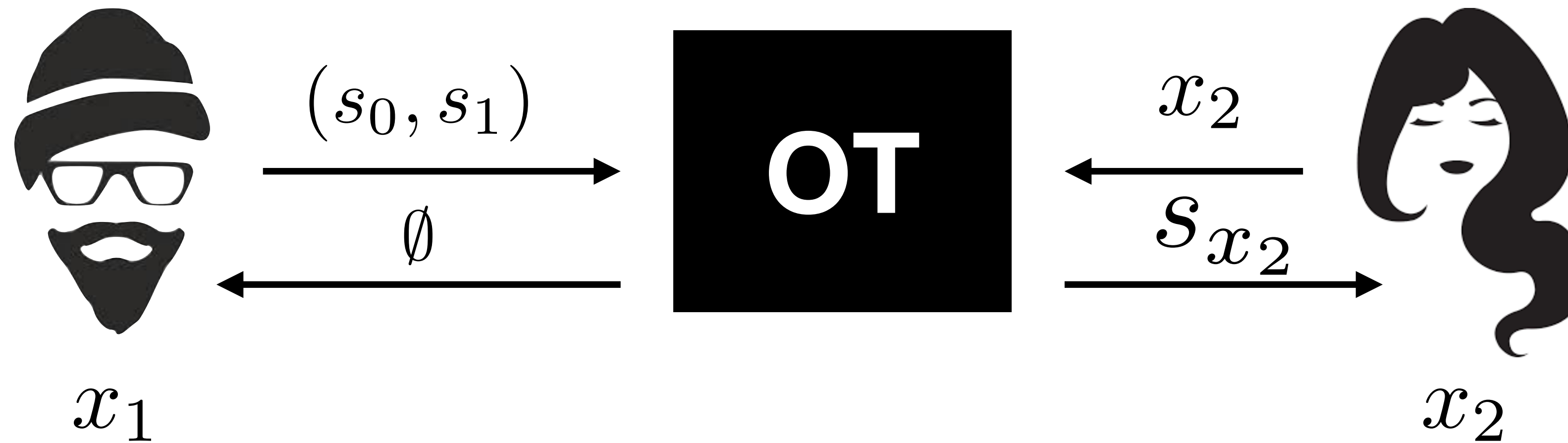
$x_1$

Core idea: a secure product functionality is an oblivious transfer in disguise!



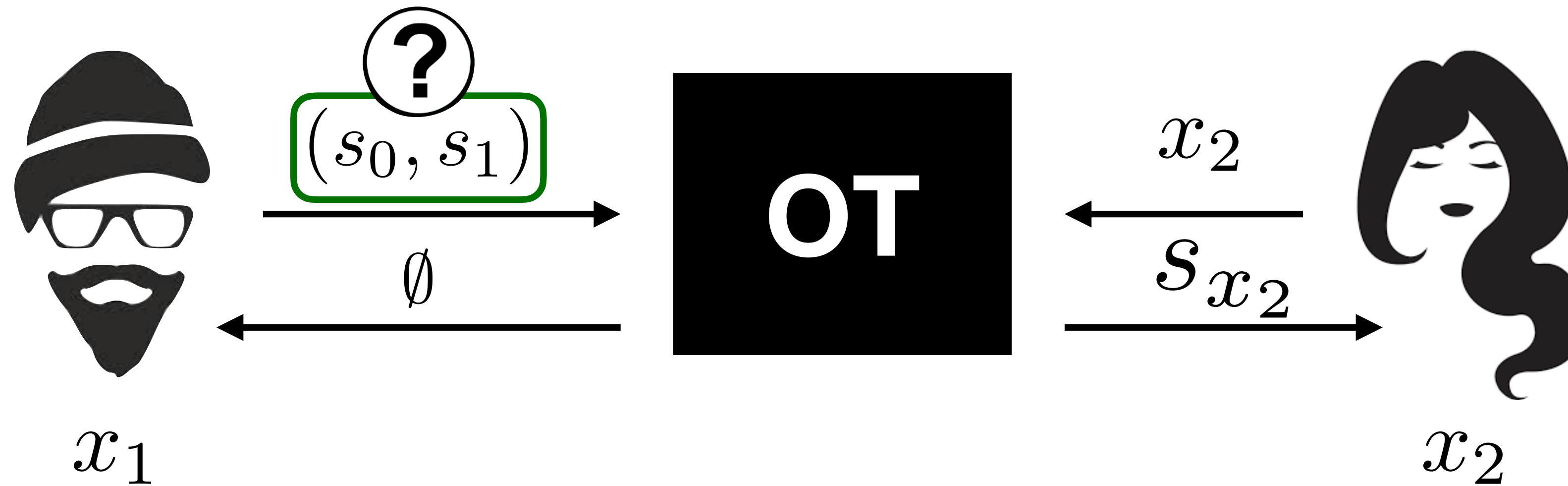
$x_2$

# Step-by Step Solution



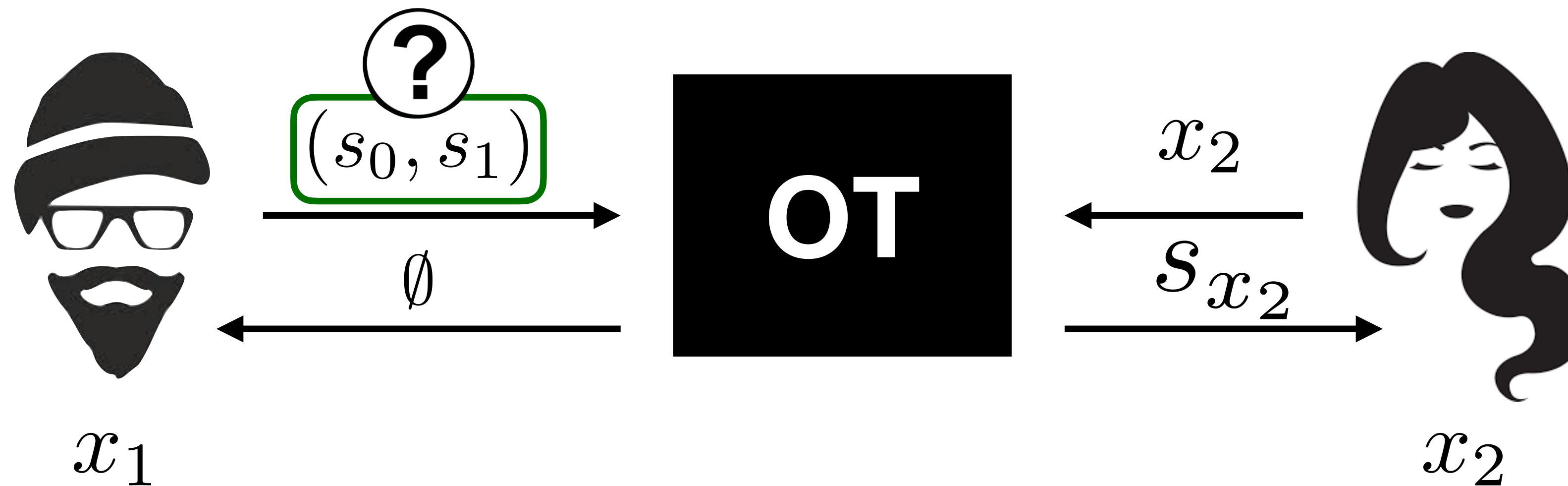
- We use an OT functionality where Alice is the receiver, and her *selection bit* is her input  $x_2$

# Step-by Step Solution



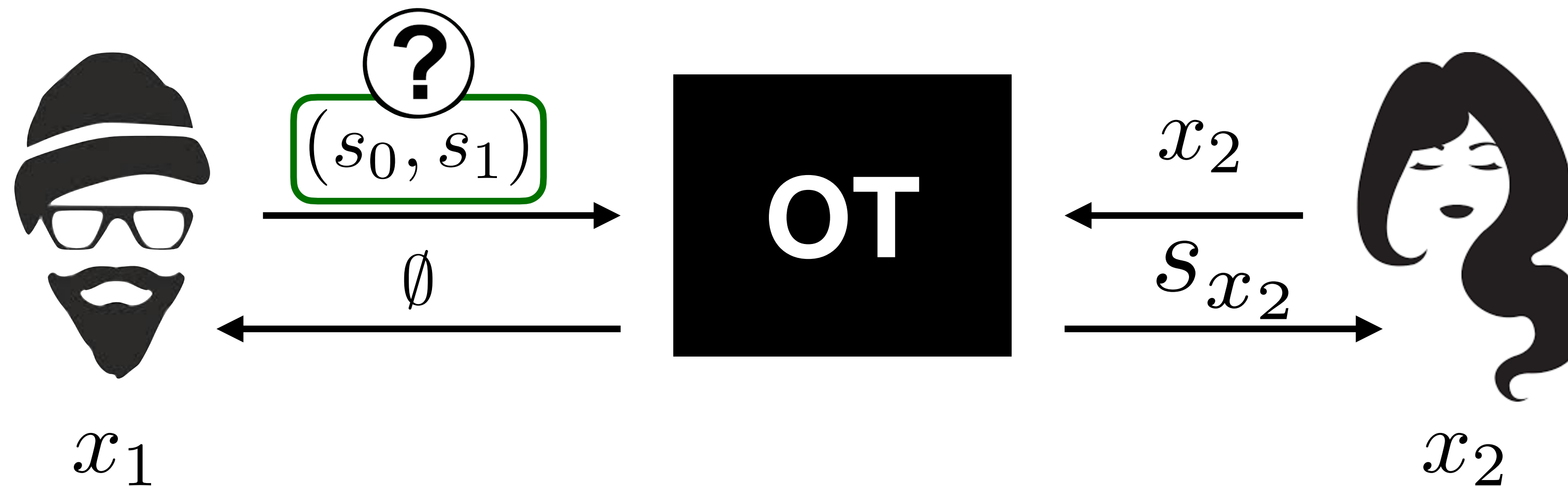
- We use an OT functionality where Alice is the receiver, and her *selection bit* is her input  $x_2$

# Step-by Step Solution



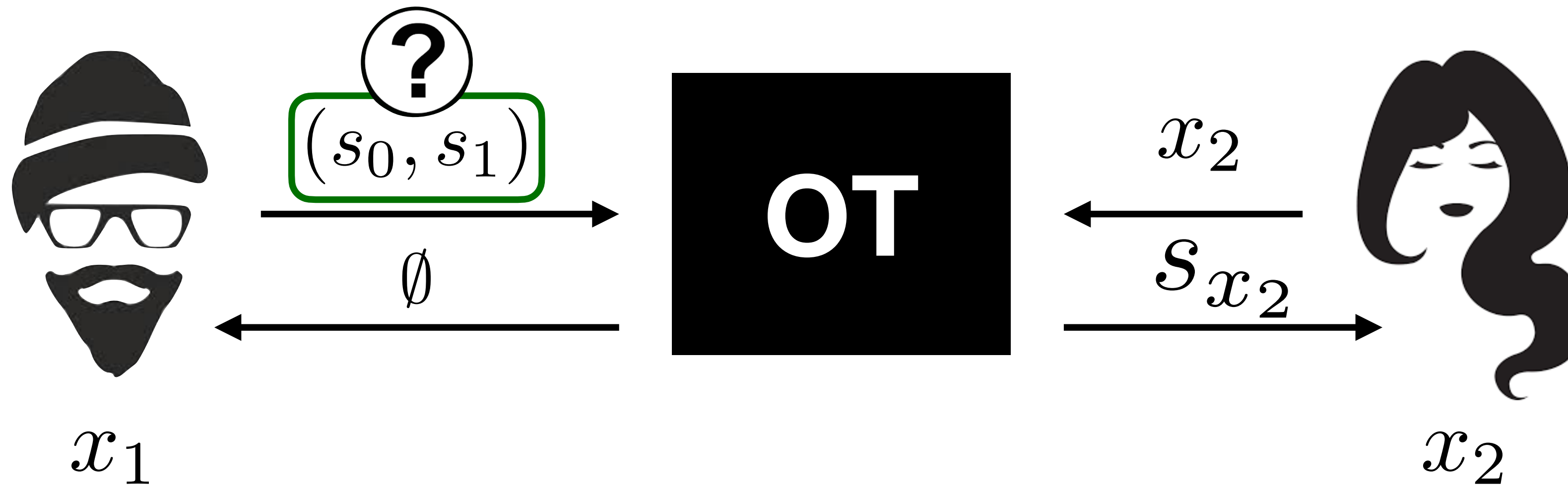
- We use an OT functionality where Alice is the receiver, and her *selection bit* is her input  $x_2$
- What should be Bob's input?

# Step-by Step Solution



- We use an OT functionality where Alice is the receiver, and her *selection bit* is her input  $x_2$
- What should be Bob's input? Let's work out the equation:

# Step-by Step Solution

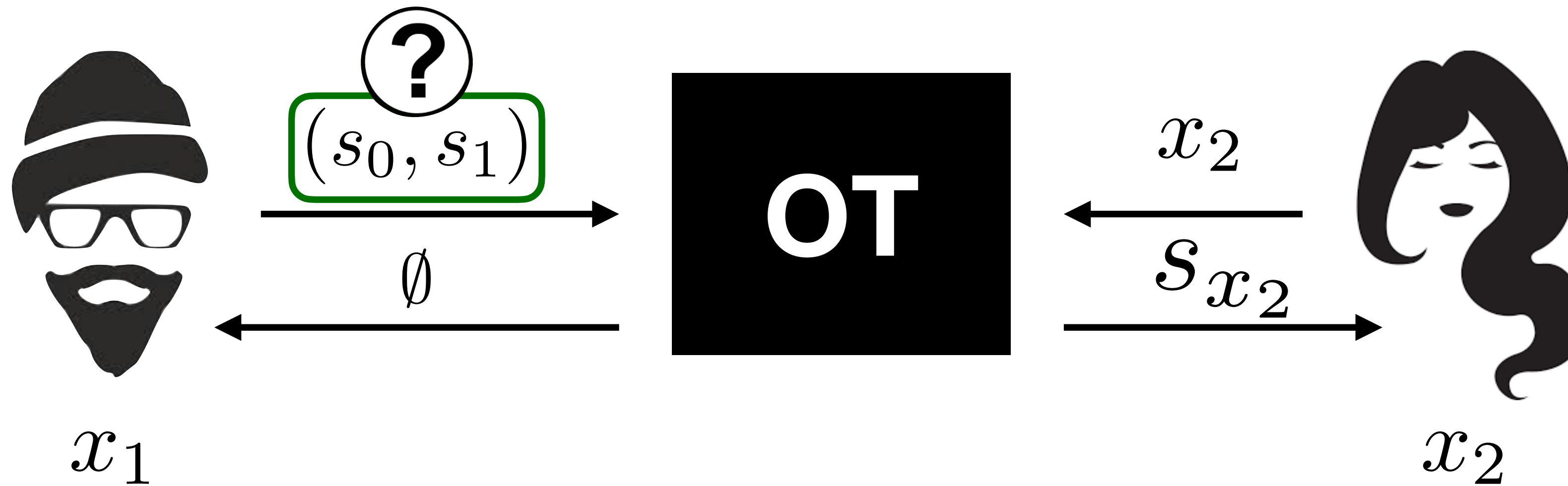


- We use an OT functionality where Alice is the receiver, and her *selection bit* is her input  $x_2$
- What should be Bob's input? Let's work out the equation:

$$\begin{aligned} s_{x_2} &= x_2 \cdot s_1 + (1 - x_2) \cdot s_0 \\ &= x_2 \cdot s_1 \oplus (1 \oplus x_2) \cdot s_0 \\ &= s_0 \oplus (s_0 \oplus s_1) \cdot x_2 \end{aligned}$$



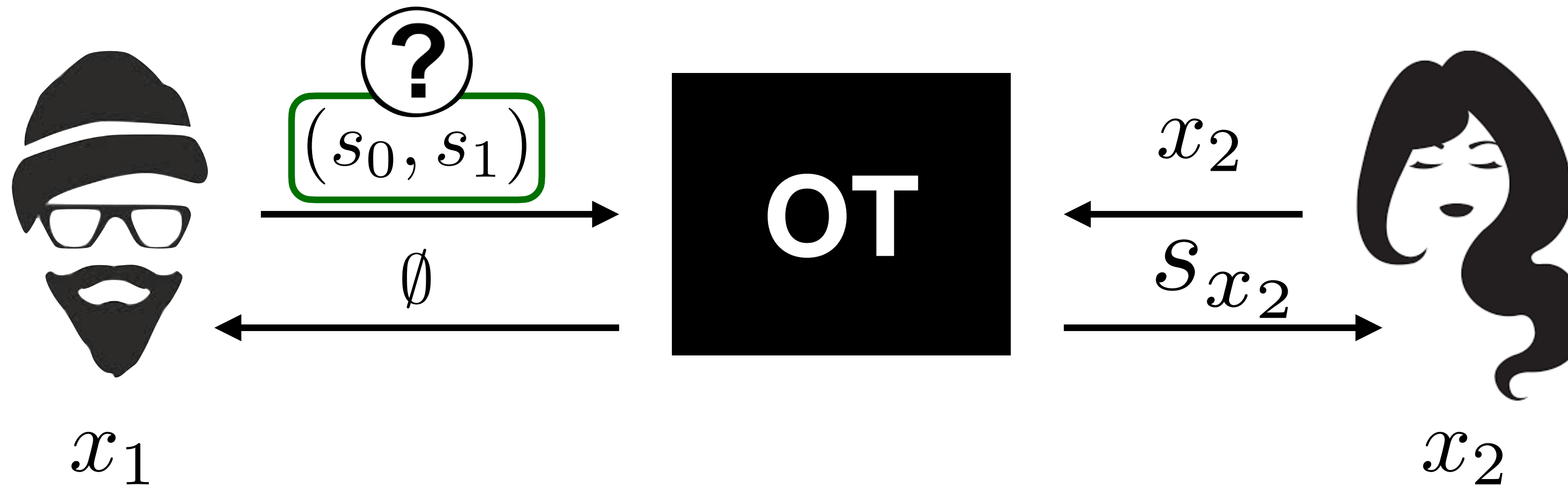
# Step-by Step Solution



- We use an OT functionality where Alice is the receiver, and her *selection bit* is her input  $x_2$
- What should be Bob's input? Let's work out the equation:

$$\begin{aligned} s_{x_2} &= x_2 \cdot s_1 + (1 - x_2) \cdot s_0 && \implies && s_0 \oplus s_{x_2} = (s_0 \oplus s_1) \cdot x_2 \\ &= x_2 \cdot s_1 \oplus (1 \oplus x_2) \cdot s_0 \\ &= s_0 \oplus (s_0 \oplus s_1) \cdot x_2 \end{aligned}$$

# Step-by Step Solution

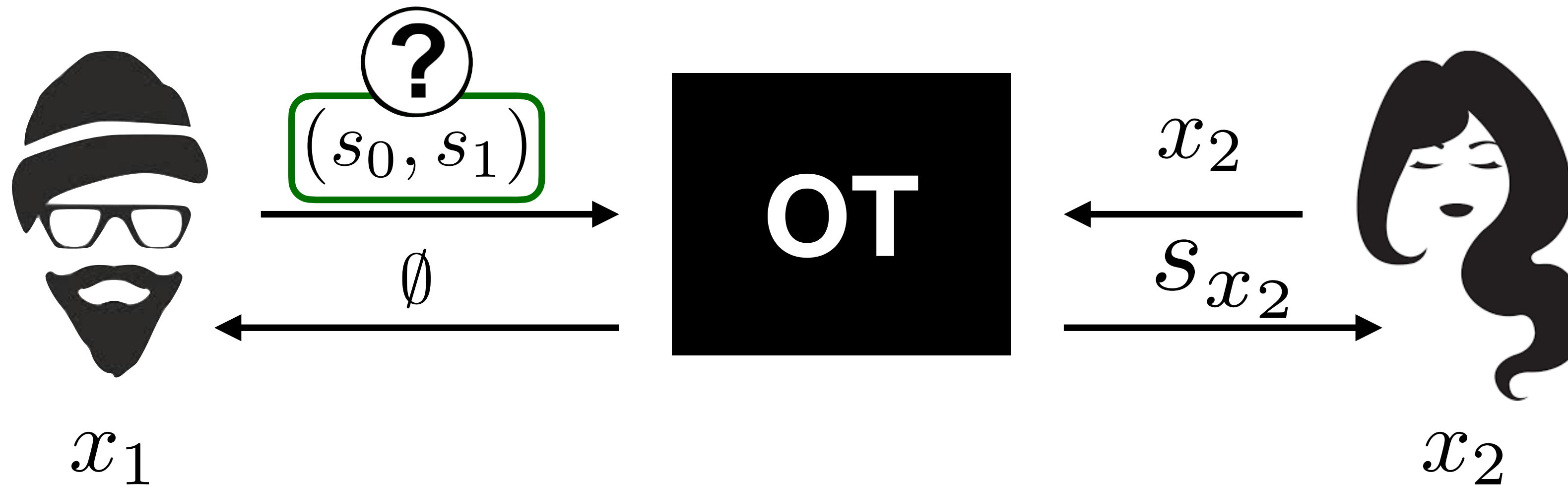


- We use an OT functionality where Alice is the receiver, and her *selection bit* is her input  $x_2$
- What should be Bob's input? Let's work out the equation:

$$\begin{aligned} s_{x_2} &= x_2 \cdot s_1 + (1 - x_2) \cdot s_0 \\ &= x_2 \cdot s_1 \oplus (1 \oplus x_2) \cdot s_0 \\ &= s_0 \oplus (s_0 \oplus s_1) \cdot x_2 \end{aligned} \quad \Rightarrow \quad \boxed{s_0} \oplus s_{x_2} = \boxed{(s_0 \oplus s_1)} \cdot x_2$$

Share of Bob      This should be  $x_1$

# Step-by Step Solution

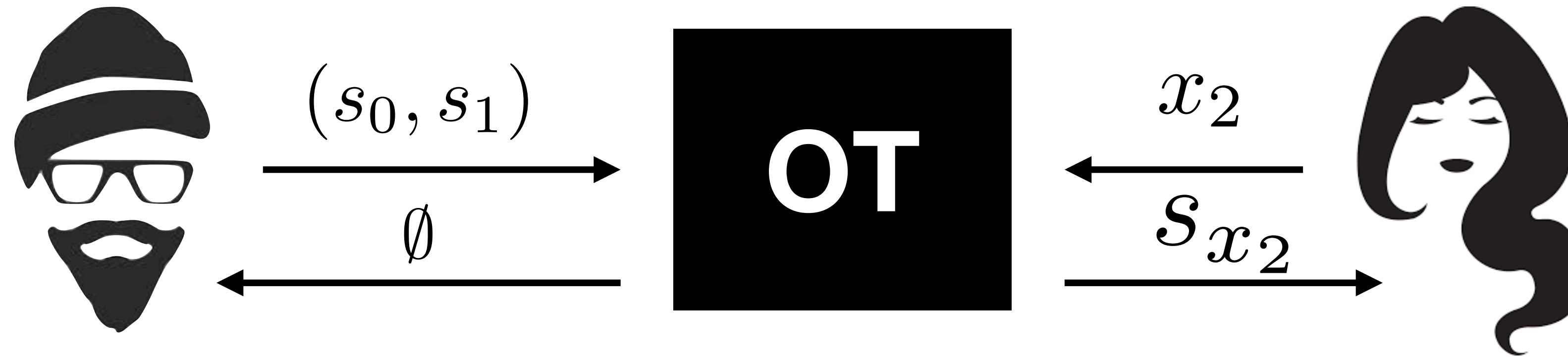


- We use an OT functionality where Alice is the receiver, and her *selection bit* is her input  $x_2$
- What should be Bob's input? Let's work out the equation:

$$\begin{aligned} s_{x_2} &= x_2 \cdot s_1 + (1 - x_2) \cdot s_0 \\ &= x_2 \cdot s_1 \oplus (1 \oplus x_2) \cdot s_0 \\ &= s_0 \oplus (s_0 \oplus s_1) \cdot x_2 \end{aligned}$$

$$\begin{aligned} &\implies \boxed{s_0} \oplus s_{x_2} = \boxed{(s_0 \oplus s_1)} \cdot x_2 \\ &\quad \text{Share of Bob} \quad \text{This should be } x_1 \\ &\implies (s_0, s_1) \text{ are } (2,2)\text{-shares of } x_1. \end{aligned}$$

# Step-by Step Solution



Shares  $x_1$  into  $(s_0, s_1)$   
(hence,  $s_0$  is uniformly random)

$x_2$

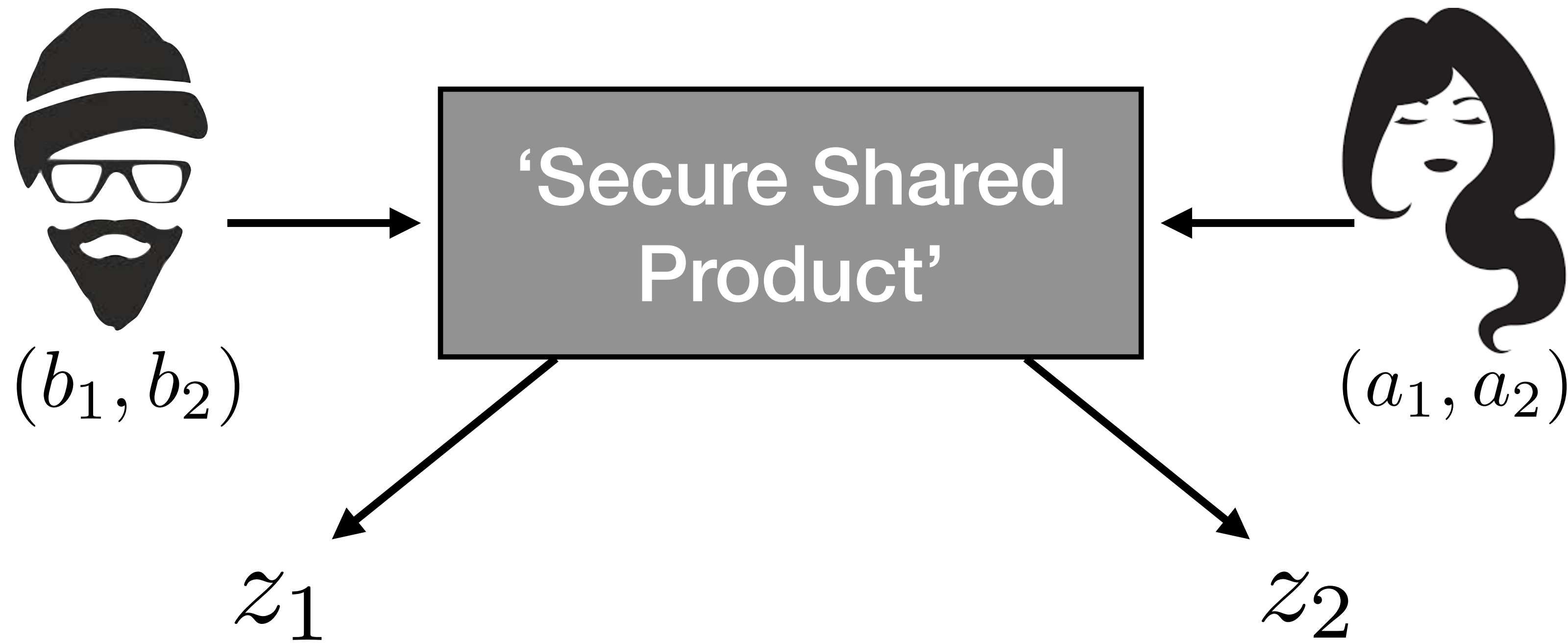
- Bob sets  $s_0$  to be his share

- Alice sets  $s_{x_2}$  to be her share

$$s_0 \oplus s_{x_2} = x_1 \cdot x_2$$

# Warm-up II: Variant

This time, Alice and Bob start with *shares* of values  $(x,y)$ , and want to compute shares of the product  $x \cdot y$



$(a_1, b_1)$  are shares of  $x$

$(a_2, b_2)$  are shares of  $y$

$(z_1, z_2)$  are random shares of  $z = x \cdot y$

# Warm-up II: Variant

This time, Alice and Bob start with *shares* of values  $(x,y)$ , and want to compute shares of the product  $x \cdot y$



**Exercise II: build this protocol**

$(a_1, b_1)$  are shares of  $x$

$(a_2, b_2)$  are shares of  $y$

$(z_1, z_2)$  are random shares of  $z = x \cdot y$

# Solution



$(b_1, b_2)$



$(a_1, a_2)$

---

$$\begin{aligned}x \cdot y &= (a_1 + b_1) \cdot (a_2 + b_2) \\ &= a_1 \cdot a_2 + a_1 \cdot b_2 + a_2 \cdot b_1 + b_1 \cdot b_2\end{aligned}$$

# Solution



$(b_1, b_2)$



$(a_1, a_2)$

$$x \cdot y = (a_1 + b_1) \cdot (a_2 + b_2)$$

$$= \boxed{a_1 \cdot a_2} + \boxed{a_1 \cdot b_2} + \boxed{a_2 \cdot b_1} + \boxed{b_1 \cdot b_2}$$

Value known to Alice



Value known to Bob

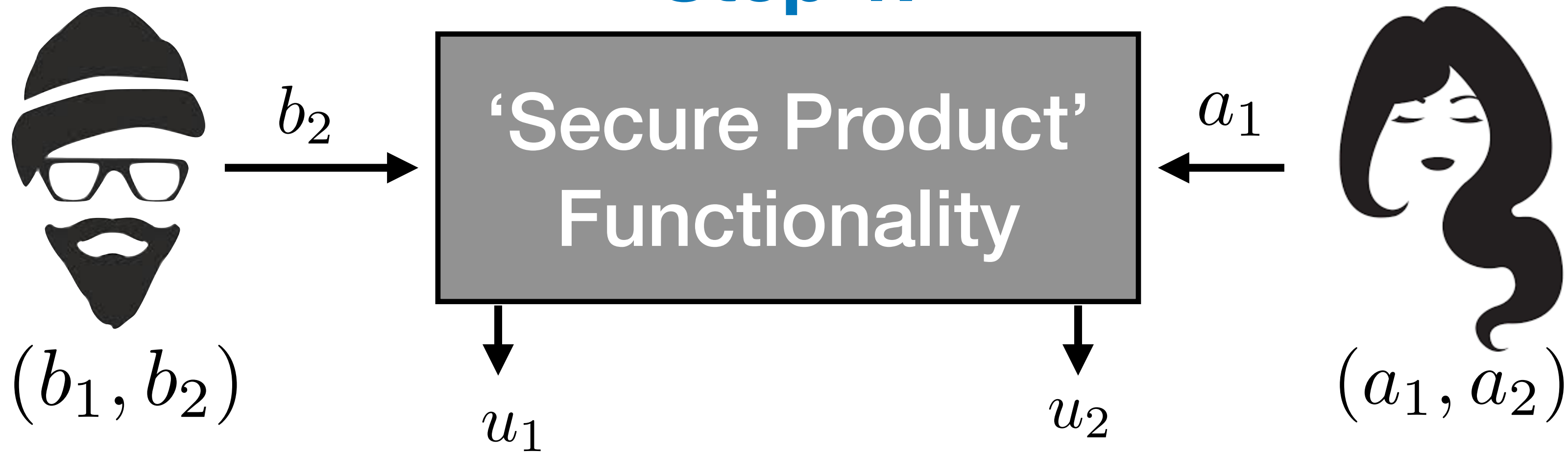


Each of these values is the product of a value known to Alice and a value known to Bob



# Solution

## Step 1:



$$x \cdot y = (a_1 + b_1) \cdot (a_2 + b_2)$$


$$= \boxed{a_1 \cdot a_2} + \boxed{a_1 \cdot b_2} + \boxed{a_2 \cdot b_1} + \boxed{b_1 \cdot b_2}$$

Value known to Alice

Value known to Bob

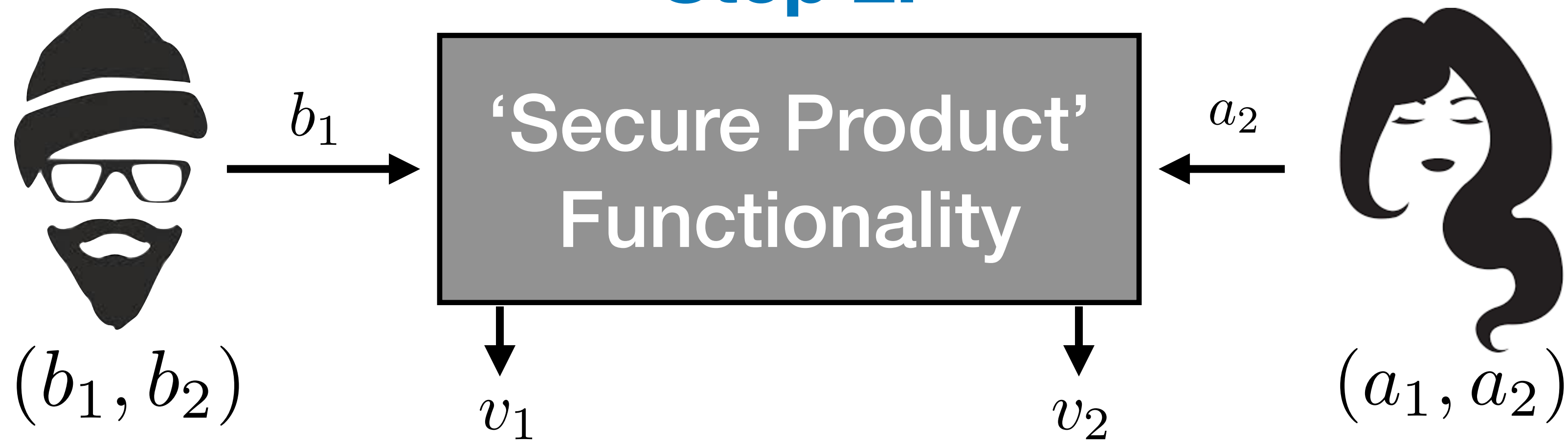
Each of these values is the product of a value known to Alice and a value known to Bob

 :  $u_1 +$

 :  $u_2 +$

# Solution

## Step 2:




$$x \cdot y = (a_1 + b_1) \cdot (a_2 + b_2)$$


$$= \boxed{a_1 \cdot a_2} + \boxed{a_1 \cdot b_2} + \boxed{a_2 \cdot b_1} + \boxed{b_1 \cdot b_2}$$

Value known to Alice

Value known to Bob

Each of these values is the product of a value known to Alice and a value known to Bob

 :  $u_1 + v_1 +$

 :  $u_2 + v_2 +$

# Solution

## Step 3:



$(b_1, b_2)$

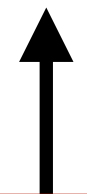
Local computation.



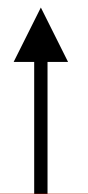
$(a_1, a_2)$

$$\begin{aligned}x \cdot y &= (a_1 + b_1) \cdot (a_2 + b_2) \\ &= \boxed{a_1 \cdot a_2} + \boxed{a_1 \cdot b_2} + \boxed{a_2 \cdot b_1} + \boxed{b_1 \cdot b_2}\end{aligned}$$

Value known to Alice



Value known to Bob



Each of these values is the product of a value known to Alice and a value known to Bob



:  $u_1 + v_1 + b_1 \cdot b_2$



:  $u_2 + v_2 + a_1 \cdot a_2$

# Solution

## Step 3:

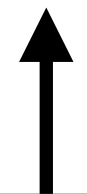
Local computation.



$$x \cdot y = (a_1 + b_1) \cdot (a_2 + b_2)$$

$$= \boxed{a_1 \cdot a_2} + \boxed{a_1 \cdot b_2} + \boxed{a_2 \cdot b_1} + \boxed{b_1 \cdot b_2}$$

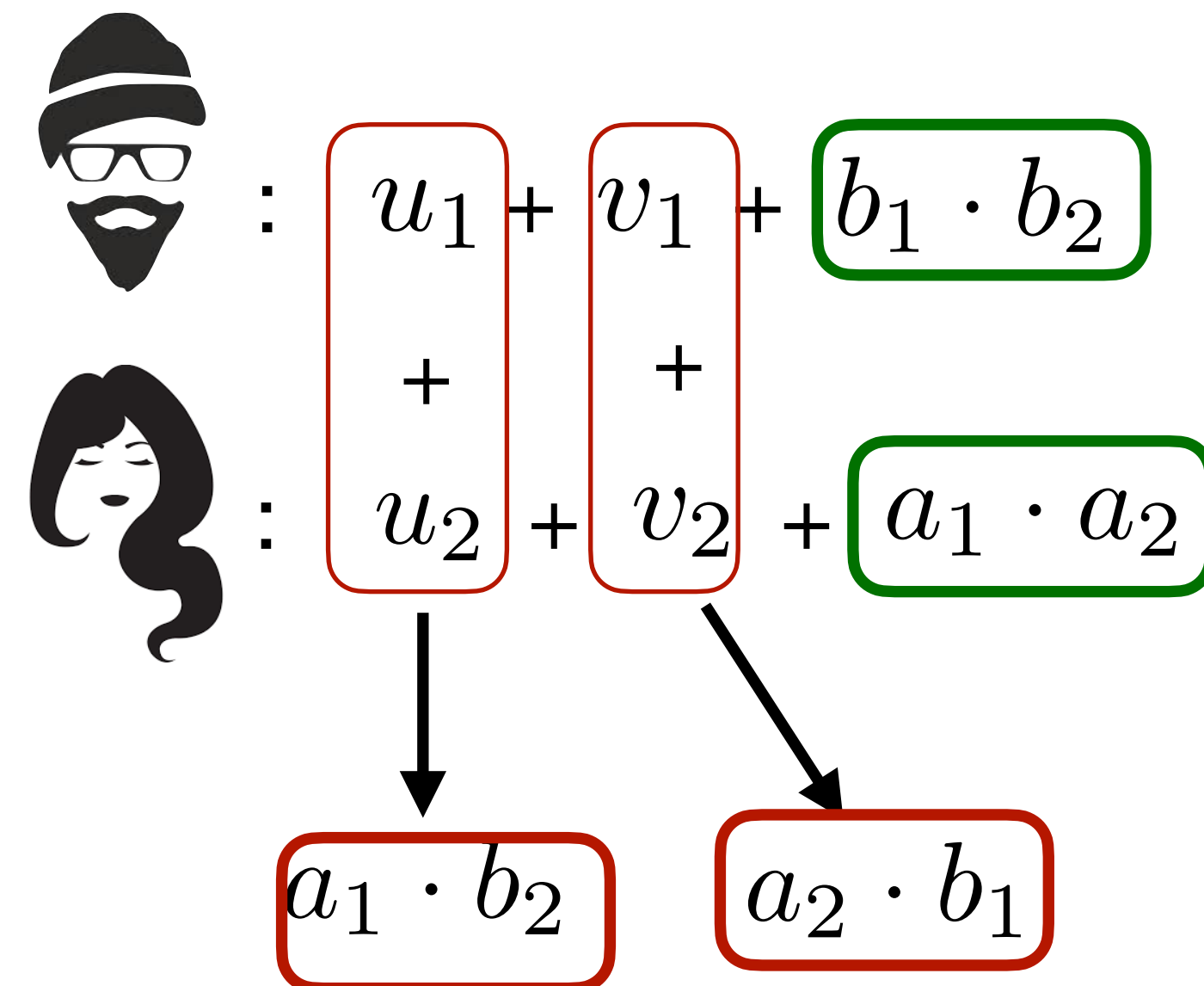
Value known to Alice



Value known to Bob

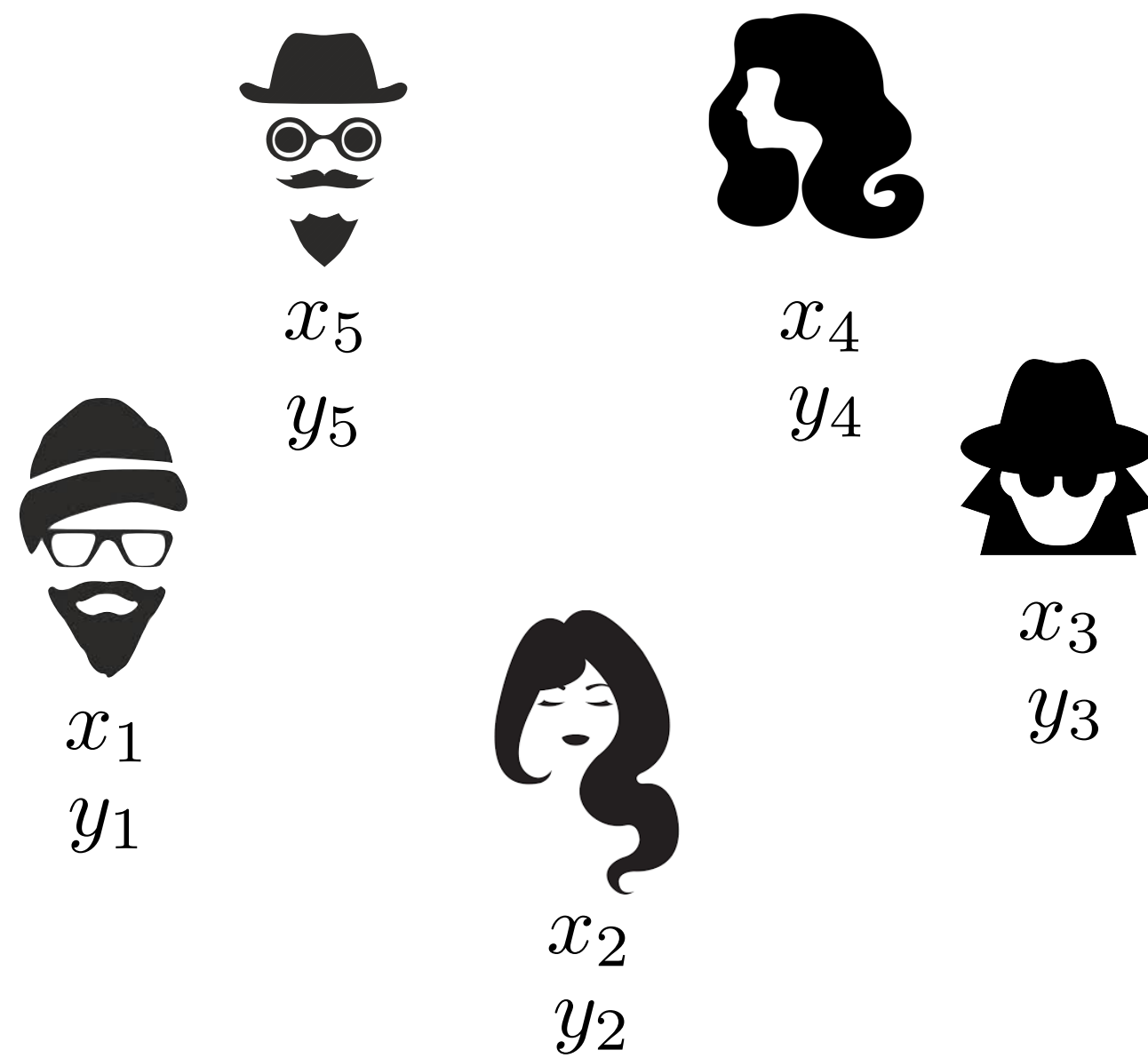


Each of these values is the product of a value known to Alice and a value known to Bob



# Warm-up III: Generalization

This time, we have  $n$  parties, holding  $(n,n)$  shares of  $x$  and  $y$ , and they should compute  $(n,n)$  shares of  $x \cdot y$



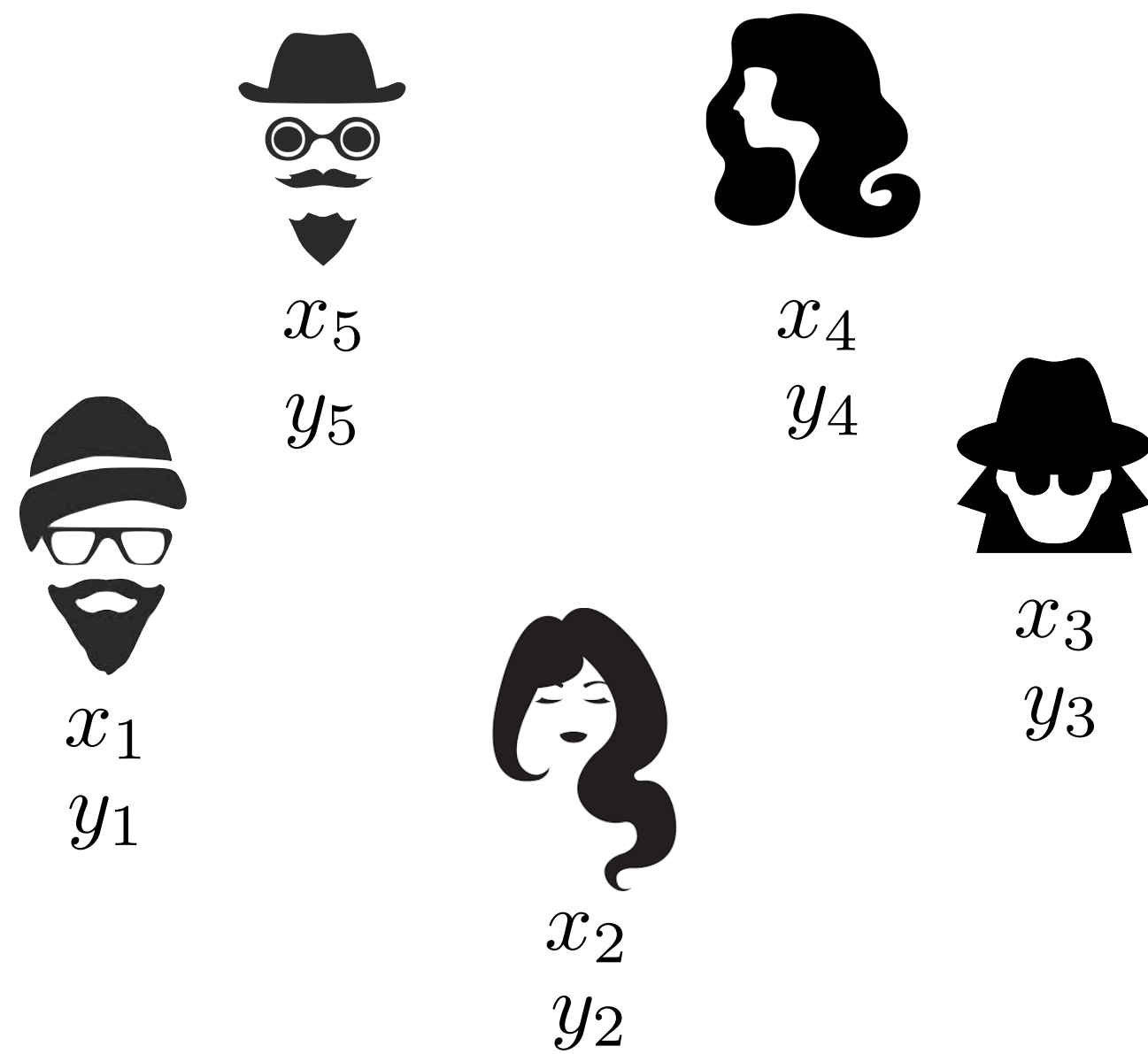
$$x = \bigoplus_{i=1}^5 x_i \qquad y = \bigoplus_{i=1}^5 y_i$$

Goal: generate uniformly random  $(z_1, \dots, z_5)$

conditioned on  $x \cdot y = \bigoplus_{i=1}^5 z_i$

# Warm-up III: Generalization

This time, we have  $n$  parties, holding  $(n,n)$  shares of  $x$  and  $y$ , and they should compute  $(n,n)$  shares of  $x \cdot y$



$$x = \bigoplus_{i=1}^5 x_i \qquad y = \bigoplus_{i=1}^5 y_i$$

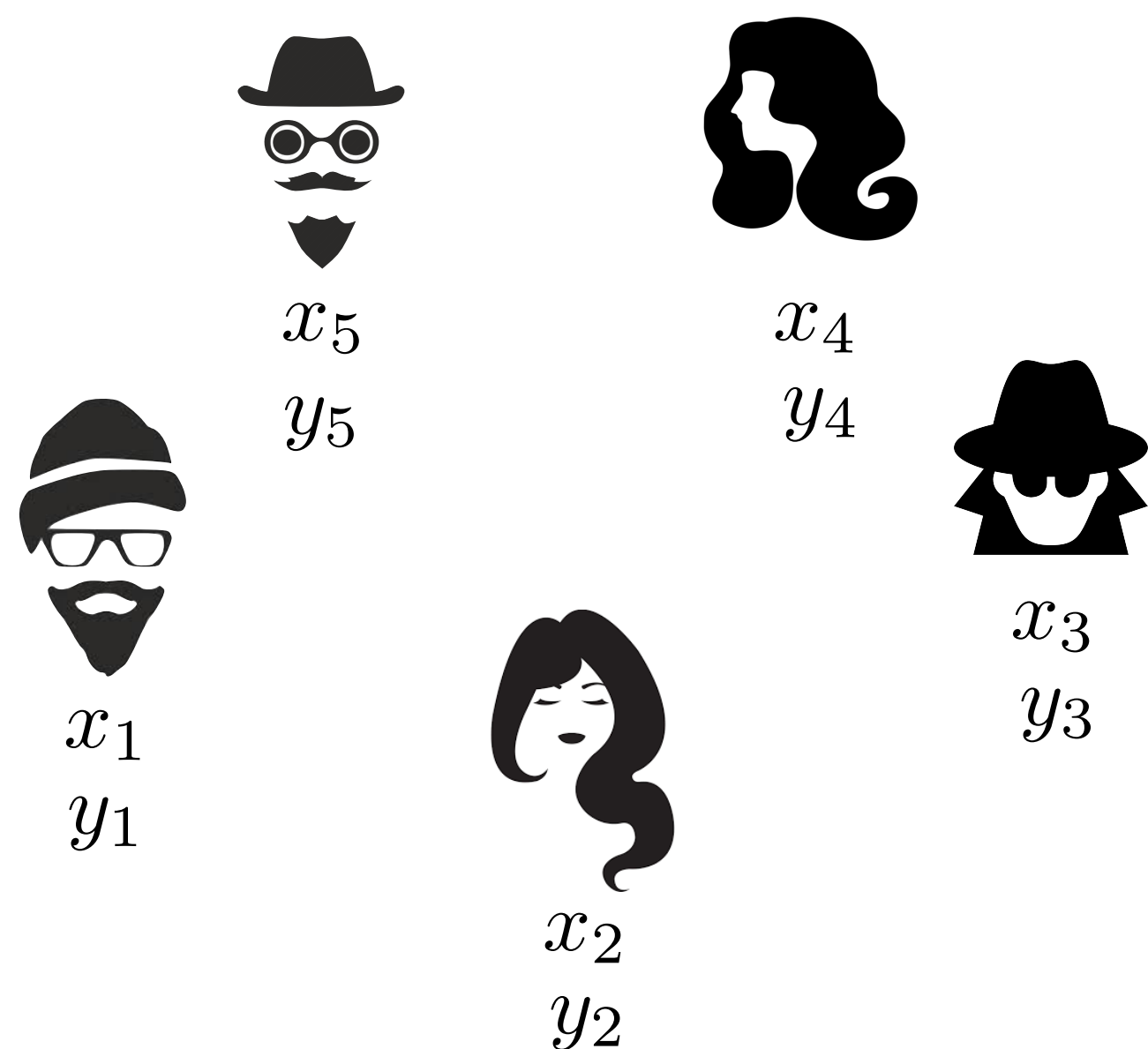
Goal: generate uniformly random  $(z_1, \dots, z_5)$

conditioned on  $x \cdot y = \bigoplus_{i=1}^5 z_i$

**Exercise III - should be easy**

# Solution

This time, we have  $n$  parties, holding  $(n,n)$  shares of  $x$  and  $y$ , and they should compute  $(n,n)$  shares of  $x \cdot y$



$$\left( \bigoplus_{i=1}^n x_i \right) \cdot \left( \bigoplus_{i=1}^n y_i \right) = \bigoplus_{i,j} x_i \cdot y_j$$

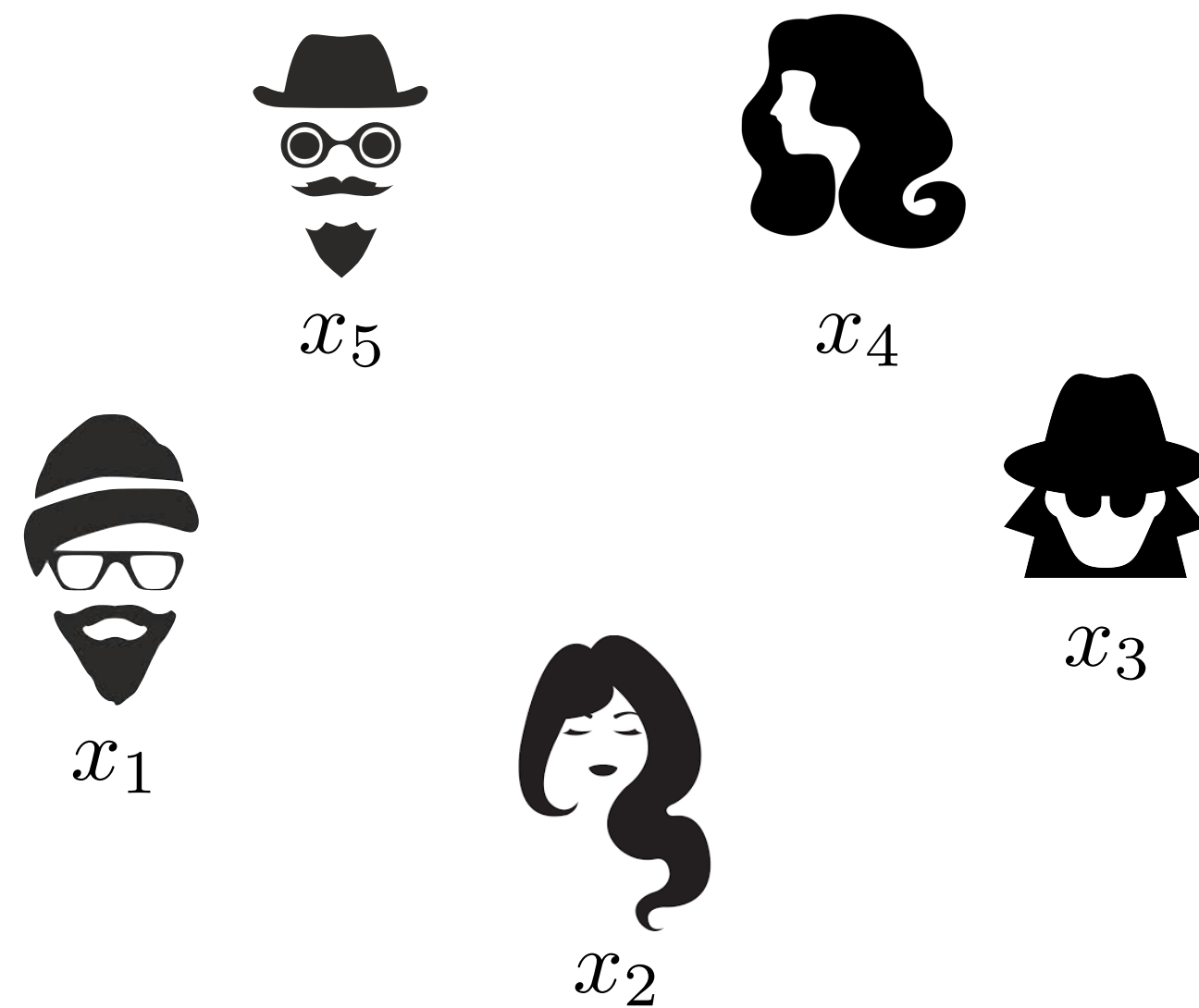
$$= \left( \bigoplus_{i \neq j} x_i \cdot y_j \right) \oplus \left( \bigoplus_i x_i \cdot y_i \right)$$

↓
Value known to player  $i$

Use a secure multiplication between players  $i$  and  $j$   
 $\Rightarrow$  needs  $n \cdot (n - 1)$  secure multiplications in total.

# Back to the GMW Protocol

## Reminder: the Model

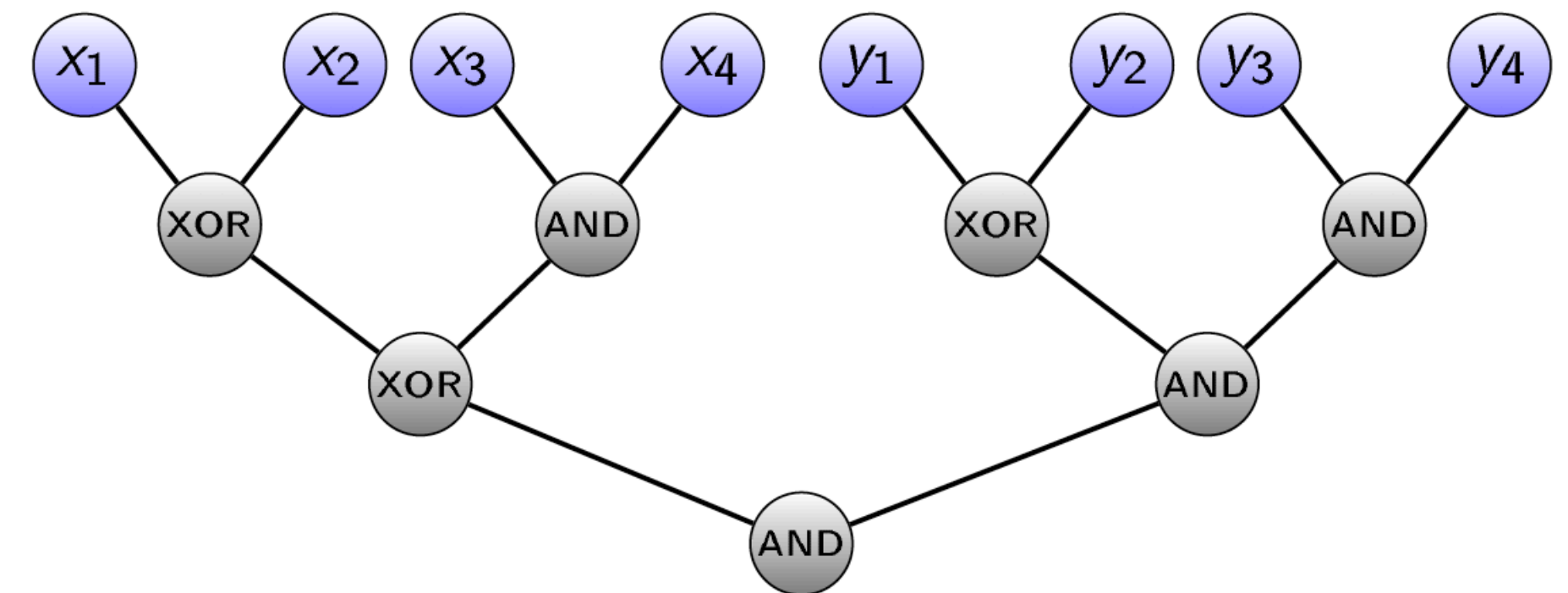


### Goal

- Public function  $f$
- All players want to get  $f(x_1, x_2, x_3, x_4, x_5)$
- No player should learn anything more

## Previously...

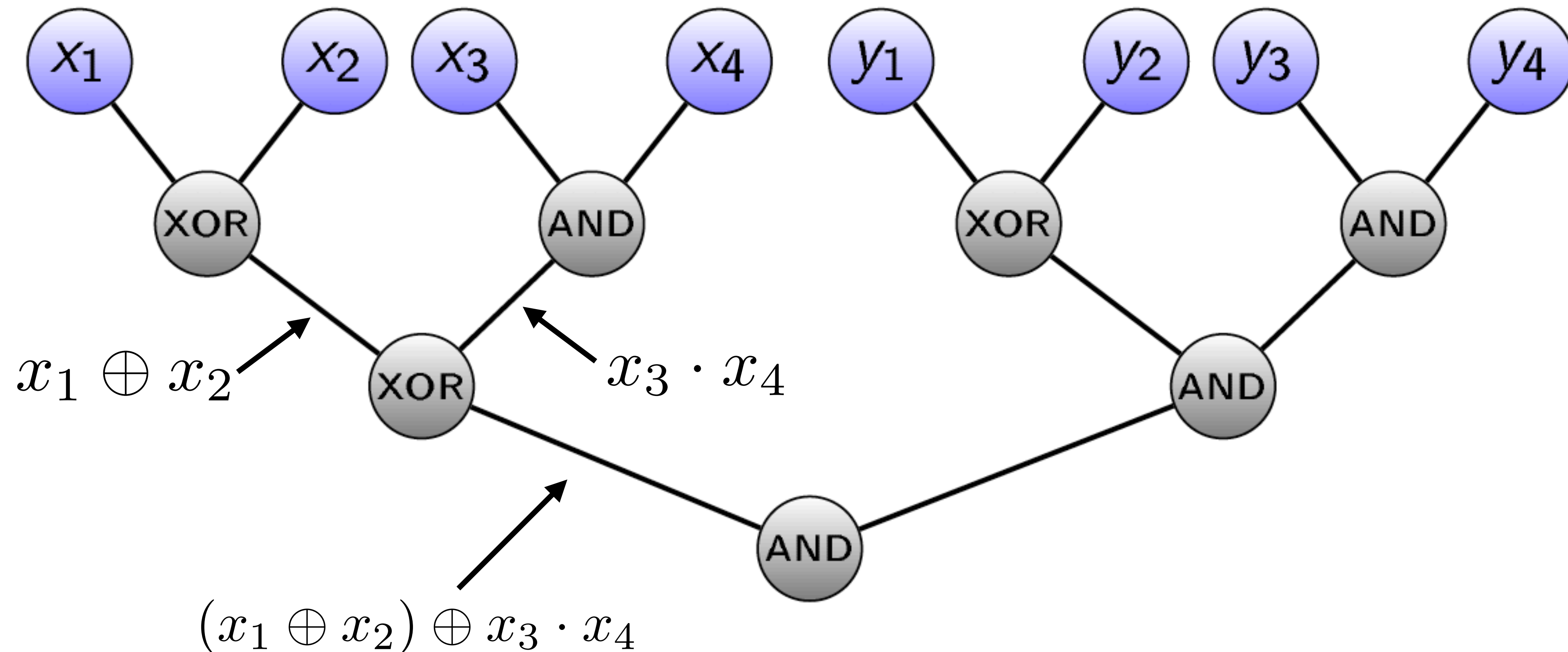
- We solved the 2-party case with garbled circuits
- Our function  $f$  was seen as a boolean circuit:





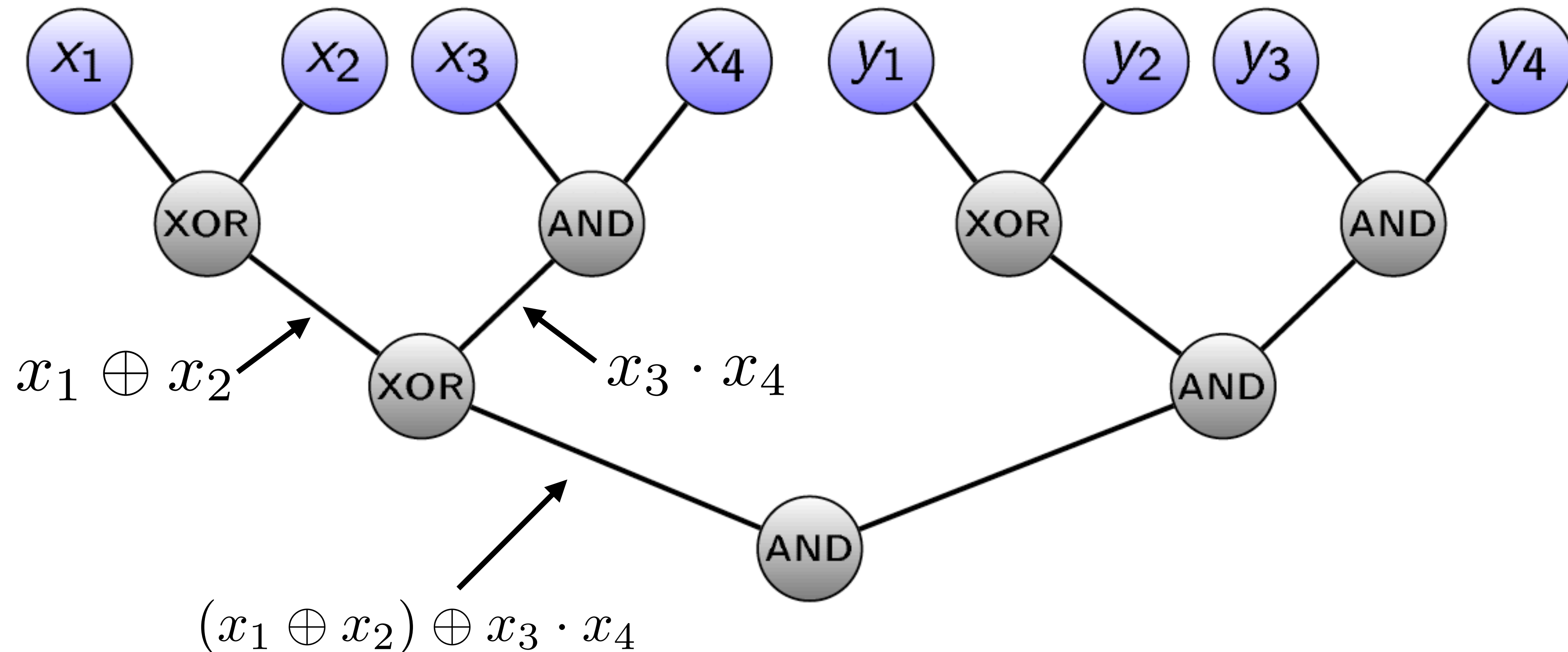
# Back to the GMW Protocol

The wires carry the intermediate values of the computation:



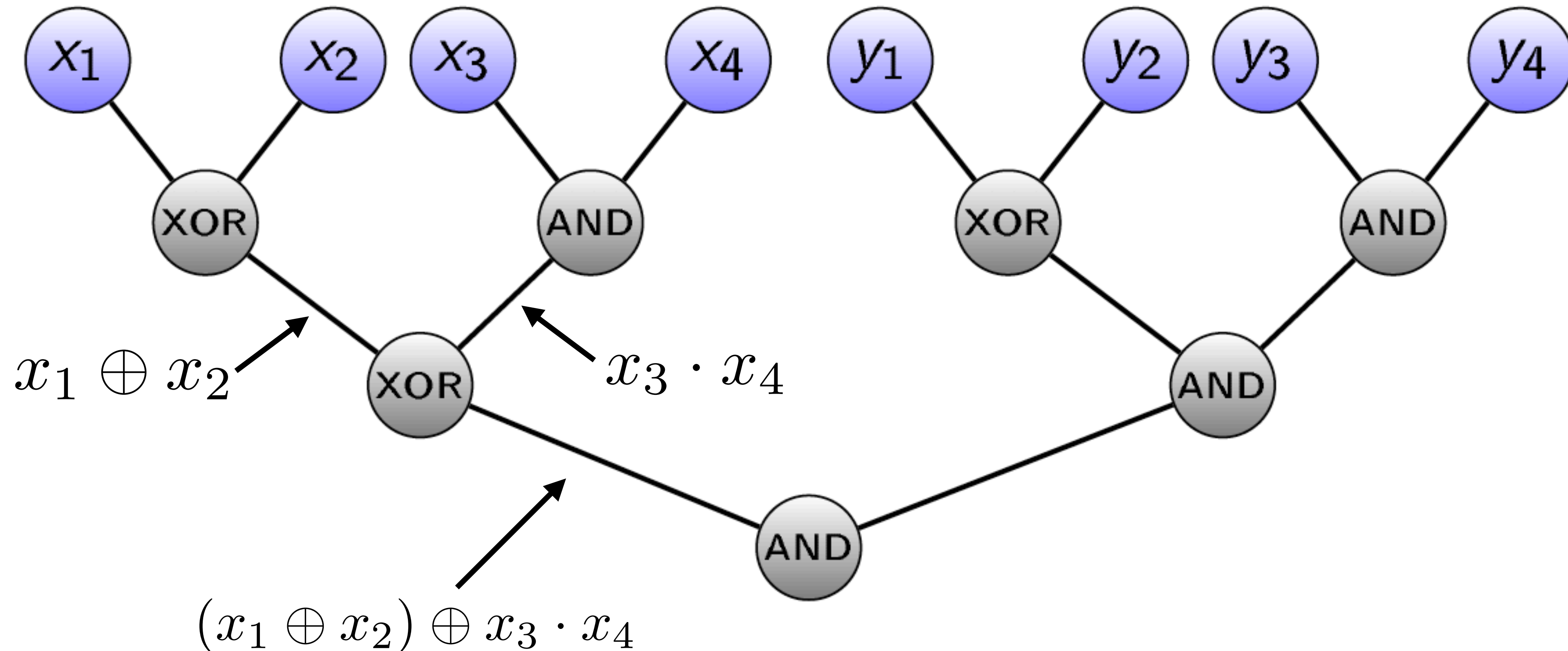
# Back to the GMW Protocol

The players will securely evaluate the boolean circuit, gate by gate. The protocol maintains the following invariant: the parties will hold  $(n,n)$ -secret shares of the values on the two input wires of the current gate, and will securely compute  $(n,n)$ -secret shares of the values on the output wire of this gate.



# Back to the GMW Protocol

Convince yourself that this works



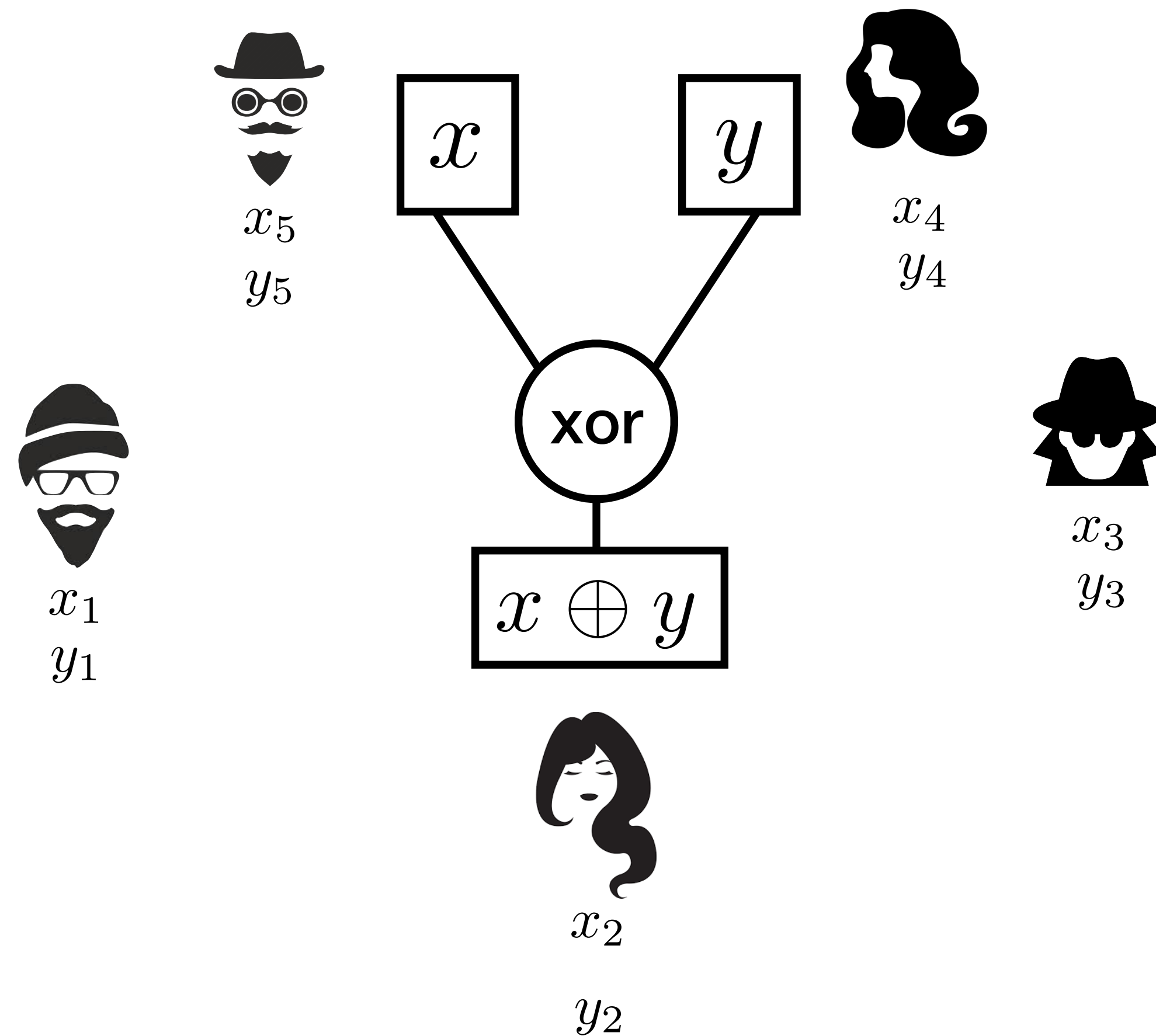
# Back to the GMW Protocol

## Evaluating a XOR gate

Inputs

$$x = \bigoplus_{i=1}^5 x_i$$

$$y = \bigoplus_{i=1}^5 y_i$$



Method

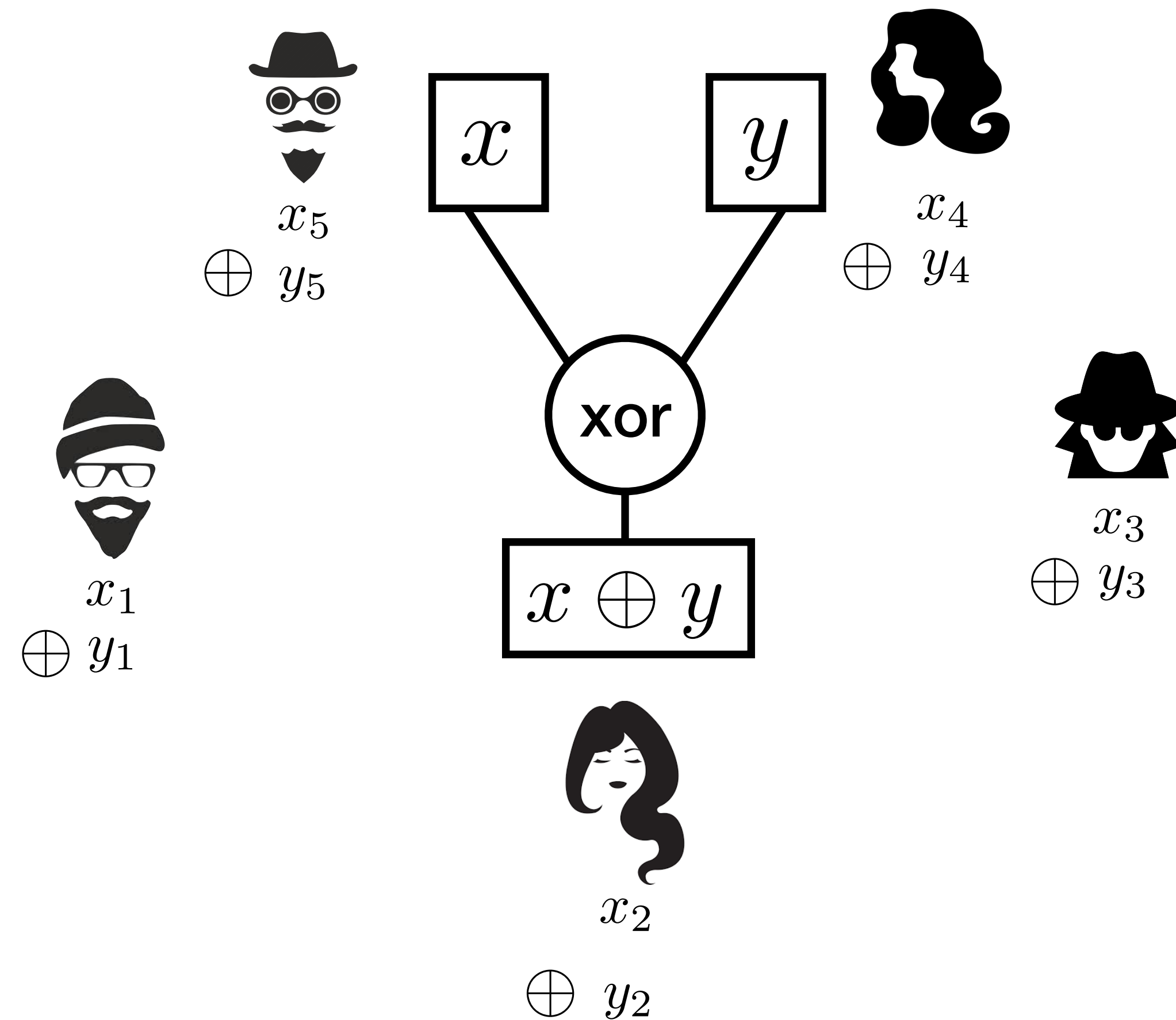
# Back to the GMW Protocol

## Evaluating a XOR gate

Inputs

$$x = \bigoplus_{i=1}^5 x_i$$

$$y = \bigoplus_{i=1}^5 y_i$$



Method

Easy: just locally xor the shares!

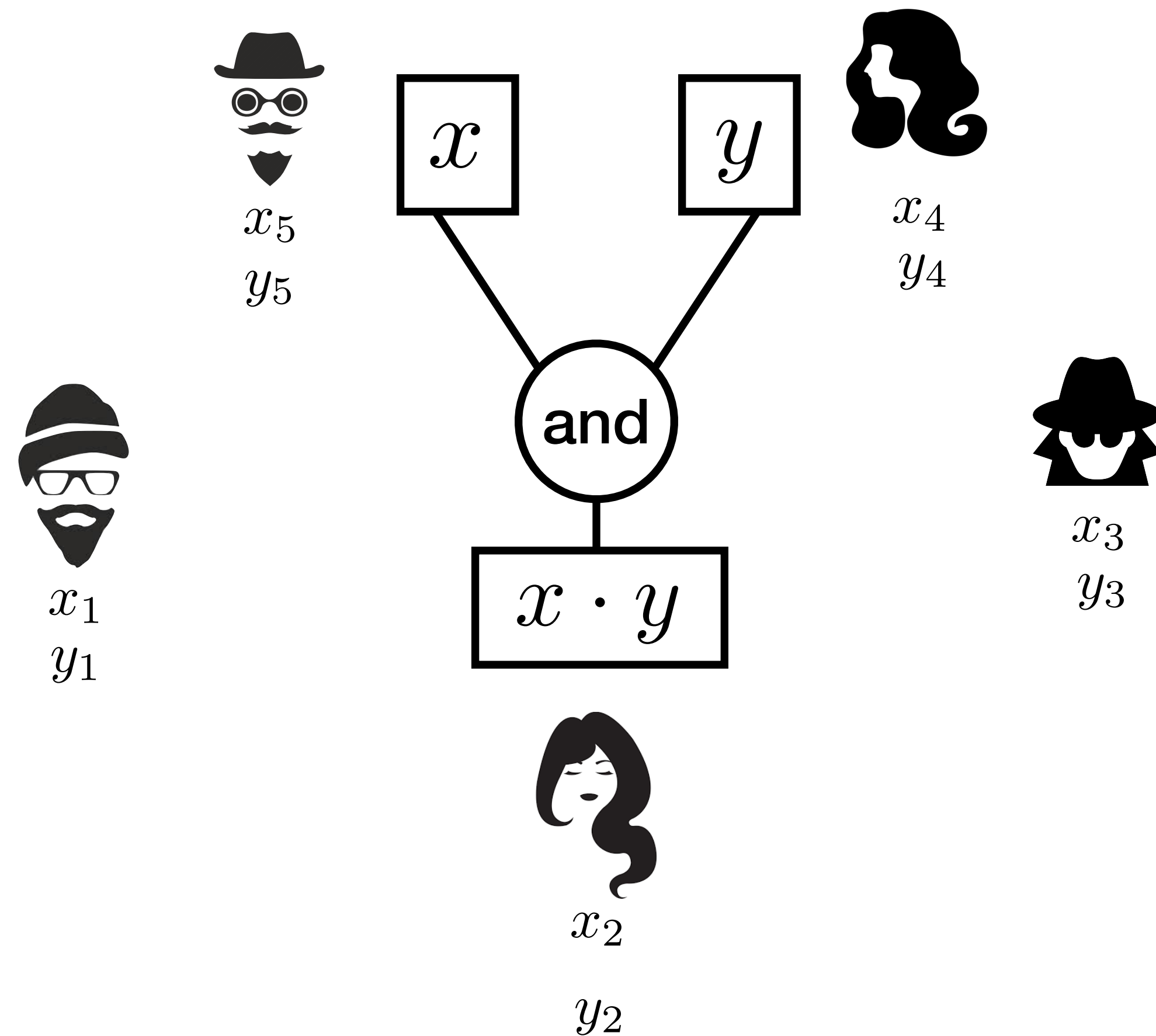
# Back to the GMW Protocol

## Evaluating a AND gate

Inputs

$$x = \bigoplus_{i=1}^5 x_i$$

$$y = \bigoplus_{i=1}^5 y_i$$



Method

We already solved it!  
(that was warm-up III)

# Wrapping Up

- Each party shares its inputs into  $n$  shares (bitwise) and sends one share / party
- For each XOR gates, the parties locally xor their shares of the input values
- For each AND gate, the parties use  $n*(n-1)$  secure multiplication protocols to reconstruct shares of the output (this necessitates  $n*(n-1)$  oblivious transfers)
- When arriving at the output wires, the parties broadcast their shares of the output value and reconstruct them.

# Comparison with Garbled Circuits

## GMW

- Function represented by a boolean circuit
- $n$  parties, use oblivious transfers for each AND gate, and local computation for each XOR gate
- Needs  $n*(n-1)*[\text{number of AND gates}]$  OTs

## Yao

- Function represented by a boolean circuit
- 2 parties, use oblivious transfers for transferring the input keys
- Garbled circuit: 4 ciphertexts / gate

Core Differences:



# Comparison with Garbled Circuits

## GMW

- Function represented by a boolean circuit
- $n$  parties, use oblivious transfers for each AND gate, and local computation for each XOR gate
- Needs  $n*(n-1)*[\text{number of AND gates}]$  OTs

## Yao

- Function represented by a boolean circuit
- 2 parties, use oblivious transfers for transferring the input keys
- Garbled circuit: 4 ciphertexts / gate

## Core Differences:

- Yao: no free XORs
- GMW: number of rounds of communication  $O(\text{circuit depth})$ , versus  $O(1)$  for Yao
- GMW: public key cryptography for each gate

# Comparison with Garbled Circuits

## GMW

- Function represented by a boolean circuit
- $n$  parties, use oblivious transfers for each AND gate, and local computation for each XOR gate
- Needs  $n*(n-1)*[\text{number of AND gates}]$  OTs

## Yao

- Function represented by a boolean circuit
- 2 parties, use oblivious transfers for transferring the input keys
- Garbled circuit: 4 ciphertexts / gate

## Core Differences:

- Yao: no free XORs  
*=> solved in [Kolesnikov-Schneider08]*
- GMW: number of rounds of communication  $O(\text{circuit depth})$ , versus  $O(1)$  for Yao  
*=> solved in [Bellare-Micali-Rogaway90]*
- GMW: public key cryptography for each gate  
*=> solved in [Ishai-Kilian-Nissim-Petrank03]*

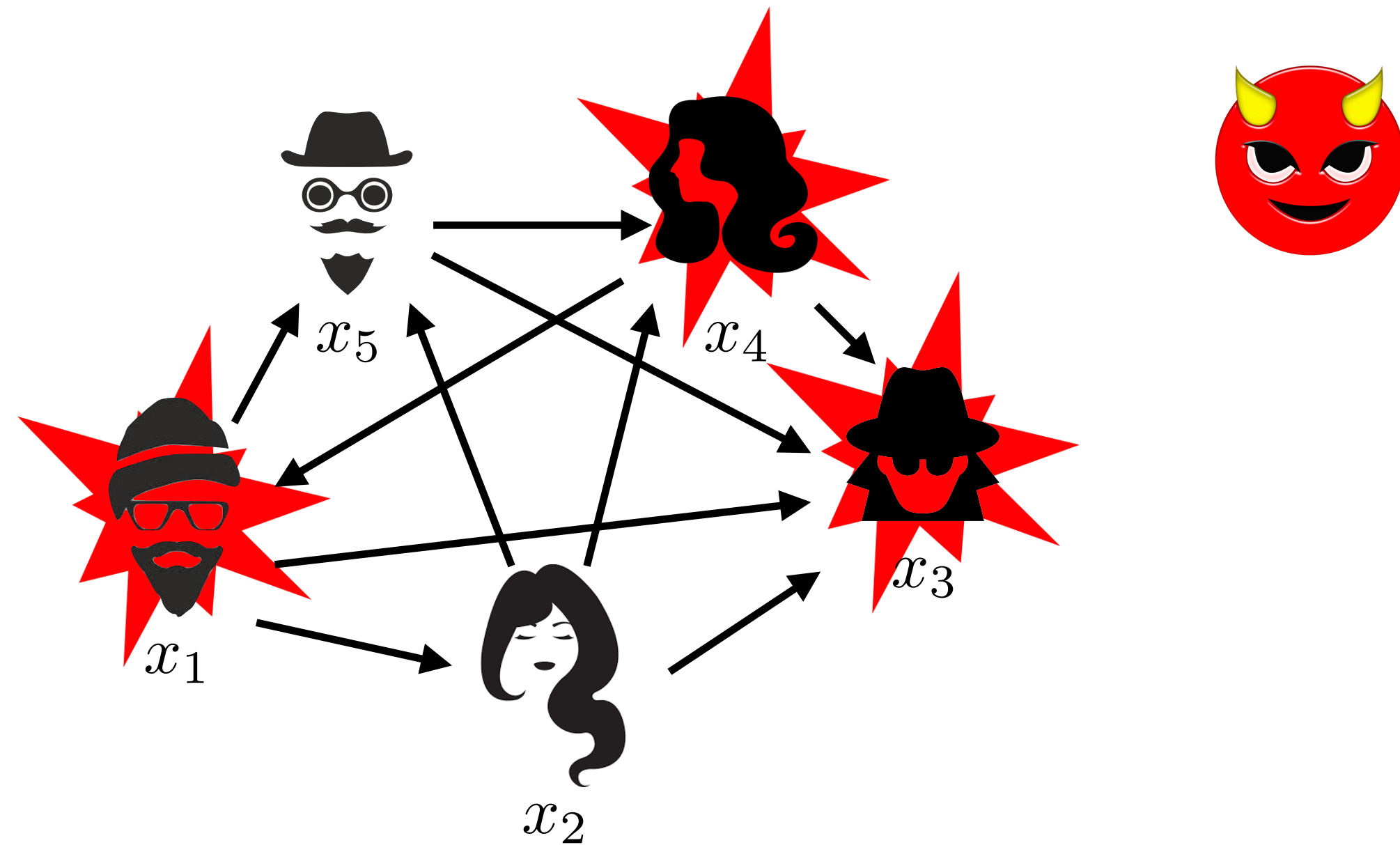
Exo: reduction randomized  $\rightarrow$  deterministic

# Secure Computation against Malicious Adversaries



# Reminder - The Model

## Real World



## *Adversarial model*

- An adversary can corrupt a subset of the players
- Two standard corruption models

## This is the model we've focused on

### 1. Honest-but-curious corruption

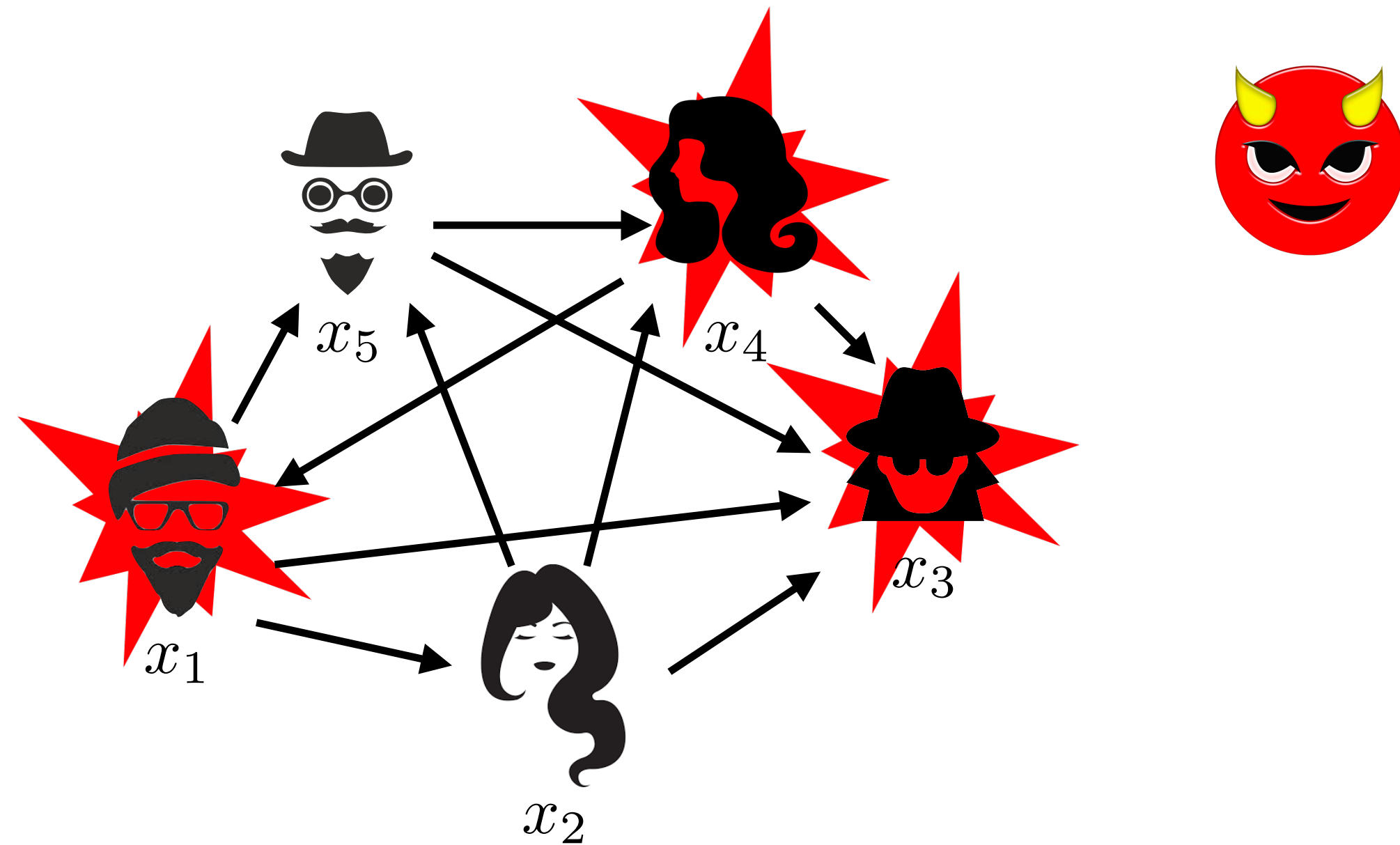
The corrupted parties follow the specification of the protocol. The adversary is passive: he tries to retrieve private information by observing the transcript.

### 2. Malicious corruption

The adversary fully control the corrupted parties, and can make them behave arbitrarily in the protocol.

# Reminder - The Model

## Real World



## *Adversarial model*

- An adversary can corrupt a subset of the players
- Two standard corruption models

### 1. Honest-but-curious corruption

The corrupted parties follow the specification of the protocol. The adversary is passive: he tries to retrieve private information by observing the transcript.

### 2. Malicious corruption

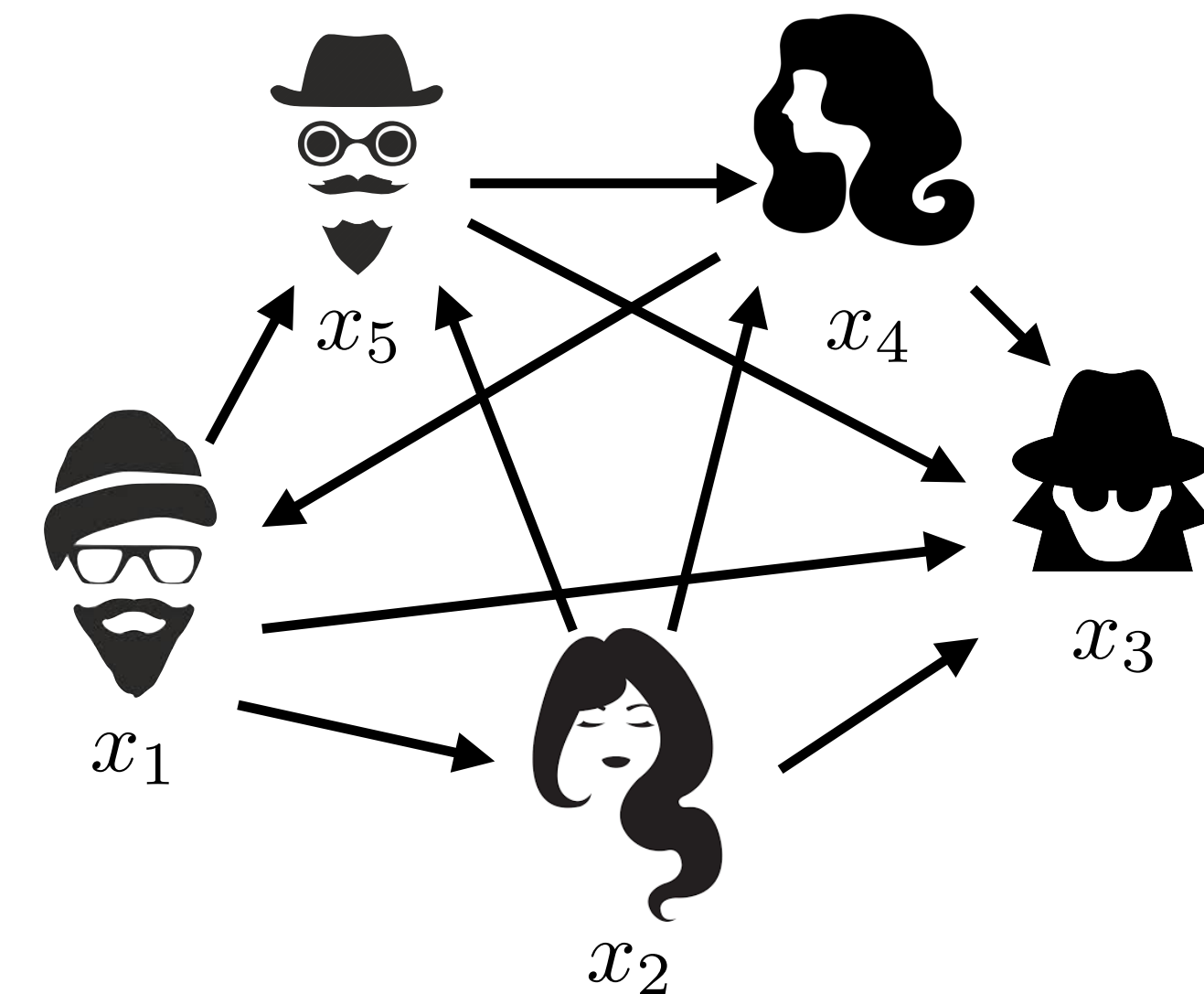
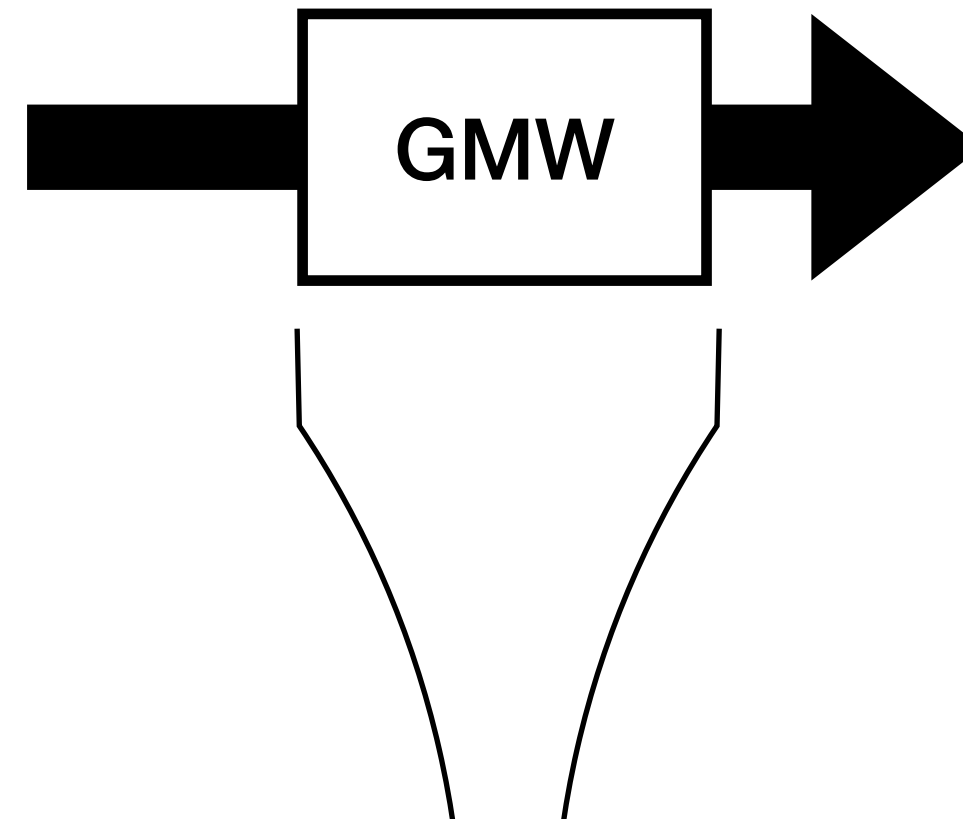
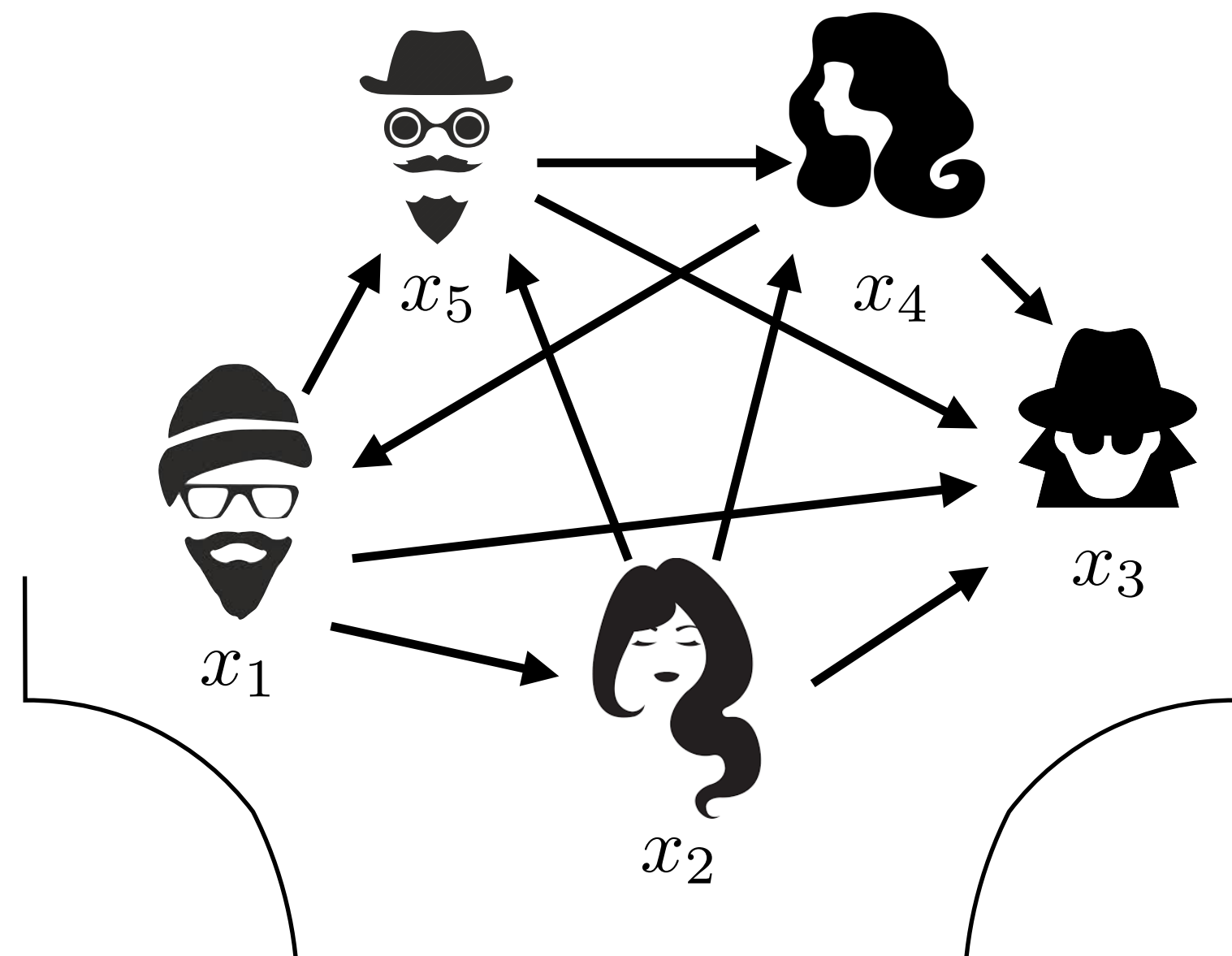
The adversary fully control the corrupted parties, and can make them behave arbitrarily in the protocol.

**Remainder of this lesson**

# Core Idea: GMW Compiler

Honest-But Curious Protocol

Malicious Protocol



We need three tools:

- 1** A semi-honest protocol  
*We already have one*

- 2** A 'commitment scheme'
- 3** A 'zero-knowledge proof system' for NP  
*=> remainder of this course*

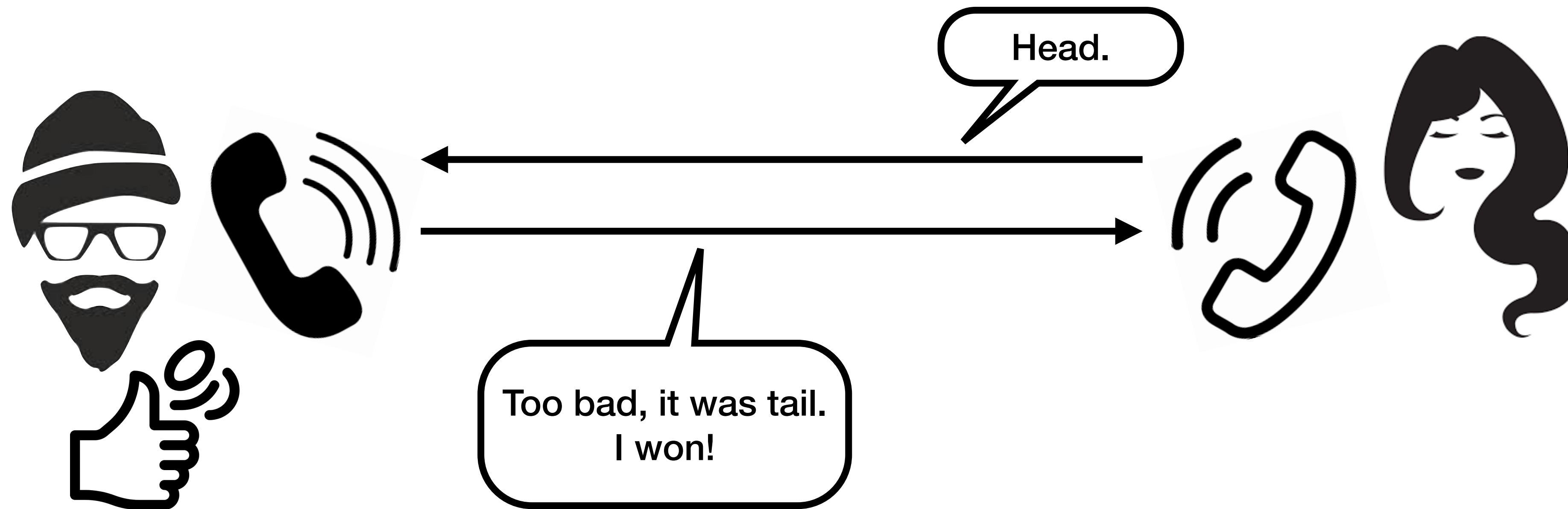
# Commitments





# Commitment Scheme: Coin-Flipping Over the Phone

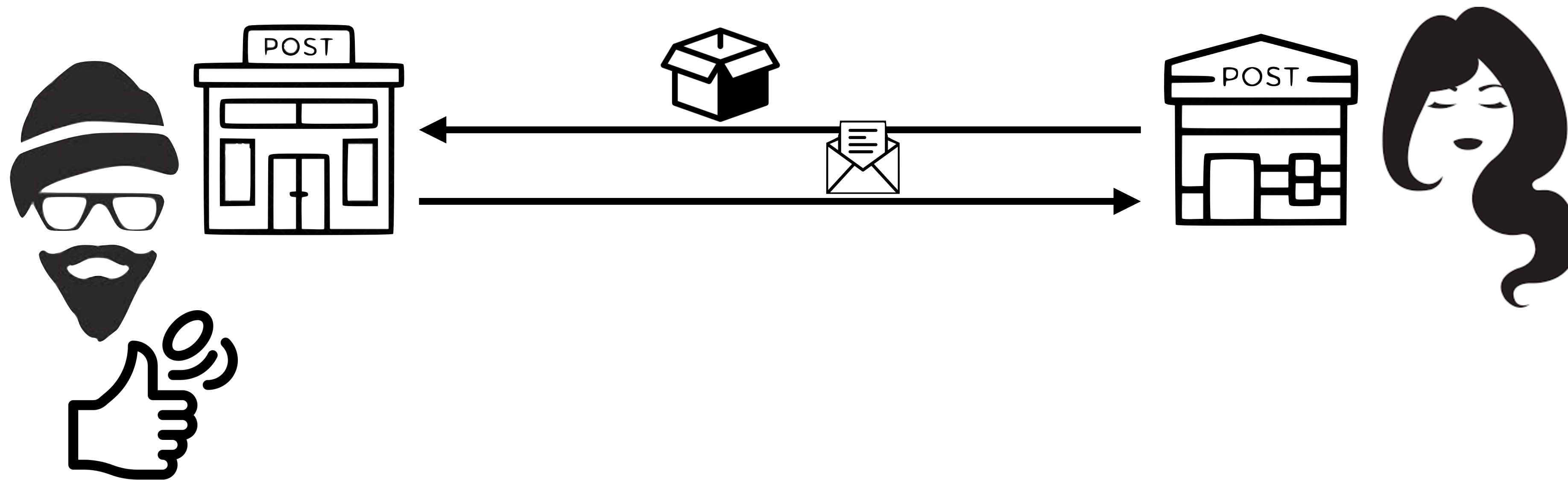
Alice and Bob want to flip a coin over the phone:



How can they prevent cheating behaviors?

# Commitment Scheme: Coin-Flipping Over the Phone

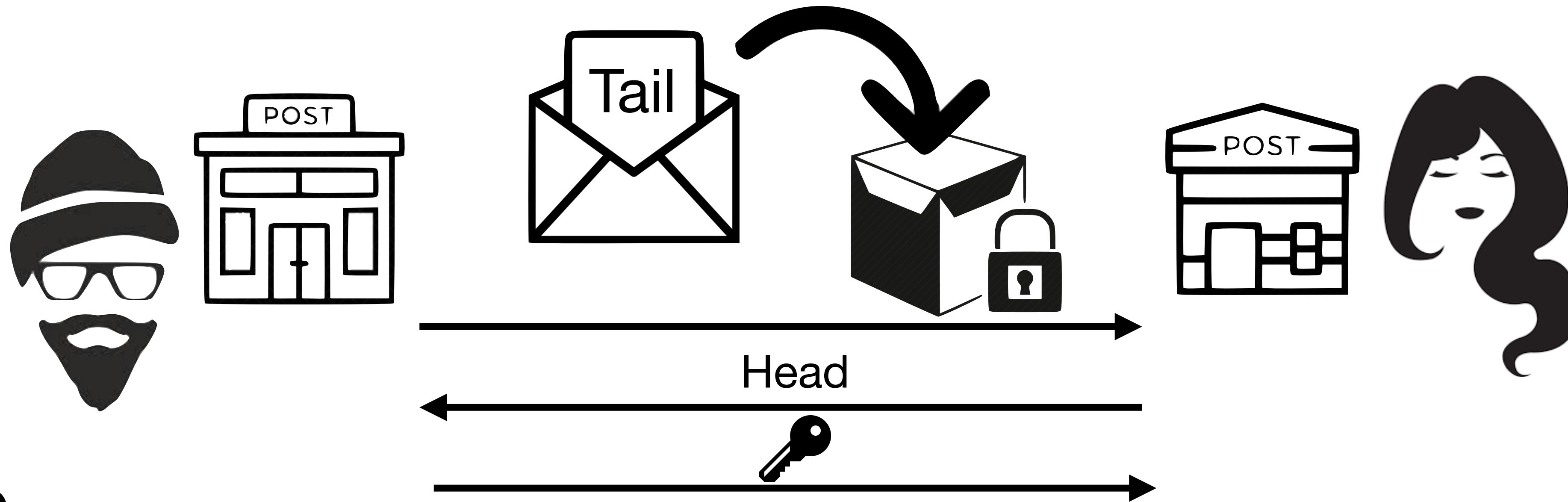
Let's first see how they could proceed using a post office



Any idea?

# Solution

Let's first see how they could proceed using a post office



**1** Bob flips a coin, writes the result on a letter, which he puts into a locked box.

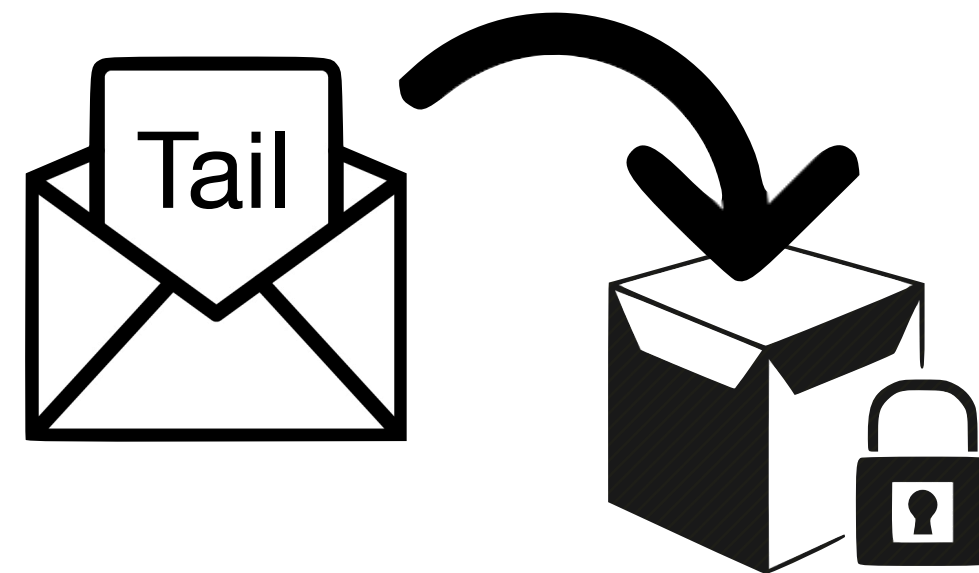
**2** Alice picks head or tail and tells it to Bob.

**3** Bob announces the result and sends the key to the box, proving that the result of the coin flip was fixed before Alice made her choice.

# Solution - Core Component

The core object used to solve the problem is a box with a message inside. Looking at the solution, there are two properties which we wanted from this object:

- When she receives a box with a message inside, Alice cannot see this message
- When Bob reveals the content of the box, he cannot lie about what was actually inside



A **commitment scheme** is a cryptographic primitive that securely realizes the above object. More precisely, a commitment scheme is a pair of algorithms (Commit, Open) such that:

- $\text{Commit}(m;r)$ , with message  $m$  and randomness  $r$ , produces a *commitment*  $c$  and an *opening*  $d$
- $\text{Verify}(c,m,d)$ , on input a commitment  $c$ , a message  $m$ , and an opening  $d$ , output 'yes' or 'no'.

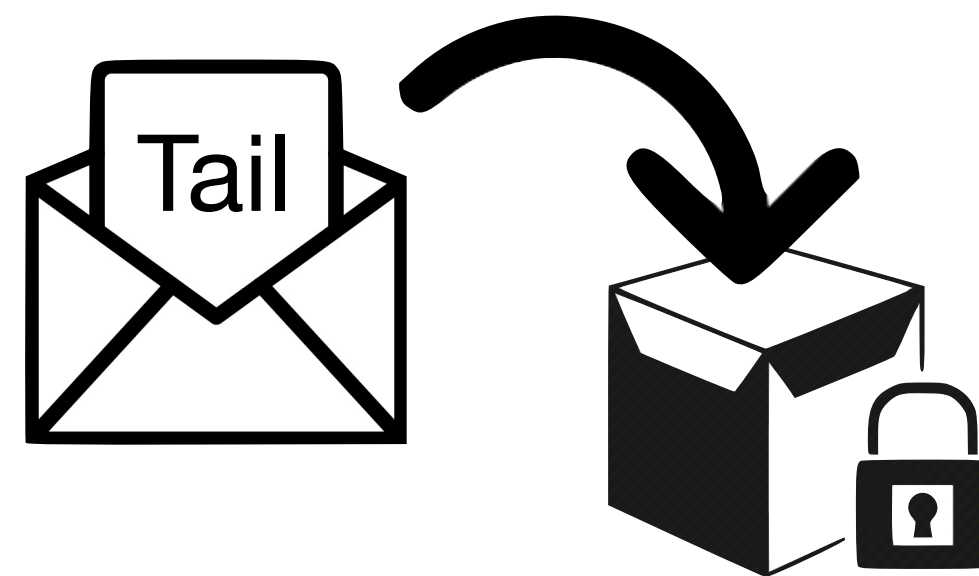
**A commitment scheme must satisfy two properties:**

- **hiding:** for all pairs  $(m, m')$ ,  $\text{Commit}(m)$  and  $\text{Commit}(m')$  are computationally indistinguishable.
- **binding:** for any commitment  $c$ , no PPT adversary can find  $(m,d)$  and  $(m',d')$  with  $m \neq m'$ , such that  $\text{Verify}(c,m,d) = \text{Verify}(c,m',d') = 1$ .

# Solution - Core Component

The core object used to solve the problem is a box with a message inside. Looking at the solution, there are two properties which we wanted from this object:

- When she receives a box with a message inside, Alice cannot see this message -> **hiding**
- When Bob reveals the content of the box, he cannot lie about what was actually inside -> **binding**



A **commitment scheme** is a cryptographic primitive that securely realizes the above object. More precisely, a commitment scheme is a pair of algorithms (Commit, Open) such that:

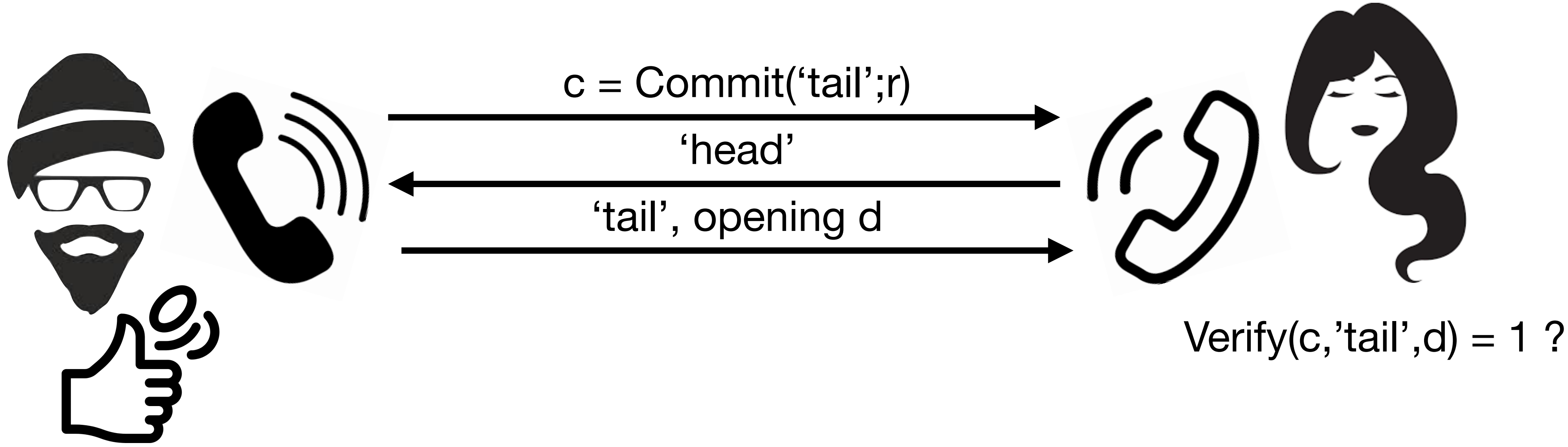
- $\text{Commit}(m;r)$ , with message  $m$  and randomness  $r$ , produces a *commitment*  $c$  and an *opening*  $d$
- $\text{Verify}(c,m,d)$ , on input a commitment  $c$ , a message  $m$ , and an opening  $d$ , output 'yes' or 'no'.

**A commitment scheme must satisfy two properties:**

- **hiding:** for all pairs  $(m, m')$ ,  $\text{Commit}(m)$  and  $\text{Commit}(m')$  are computationally indistinguishable.
- **binding:** for any commitment  $c$ , no PPT adversary can find  $(m,d)$  and  $(m',d')$  with  $m \neq m'$ , such that  $\text{Verify}(c,m,d) = \text{Verify}(c,m',d') = 1$ .

# Commitment Scheme: Coin-Flipping Over the Phone

We can finally provide a full solution to our problem:



# Commitment Scheme - Constructions

## From a pseudorandom number generator:

- This commitment scheme will be *interactive*: the Commit algorithm is actually a two-party protocol between the committer and the verifier.
- Bob selects a random  $3n$ -bit vector  $x$  and sends it to Alice
- Alice selects a random seed  $s$  and compute  $y = \text{PRG}(s)$
- Commit: if  $b = 1$ , send  $y$ , else send  $z = x \oplus y$ .
- Open: send  $s$ , Bob can then check  $z = \text{PRG}(s)$  or  $\text{PRG}(s) \oplus x$ .
- Statistical hiding: find  $(s, s')$  such that  $\text{PRG}(s) = \text{PRG}(s') \oplus x \dots$
- Computational binding:  $\text{PRG}(s) \approx \text{random}$

# Commitment Scheme - Constructions

From a collision-resistant hash function (e.g. SHA-256):

- $H$  is a function such that, for any  $y$ , it is hard to find  $x_1 \neq x_2$  with  $H(x_1) = H(x_2)$ .
  - Commit( $b$ ): pick a random string  $r$  with  $r = b \pmod{2}$ , set  $c = H(r)$ .
  - Open: reveal  $r$ . Verify( $c, b, r$ ): check that  $r = b \pmod{2}$  and  $c = H(r)$ .
- 

Security analysis:

- **Binding:** follows directly from the collision resistance.
  - **Hiding:** pretty hard - we will not do it.
- 

Note: there are constructions from pretty much anything you can think of - pseudorandom generators, AES, RSA, DDH, any standard cryptographic primitive...

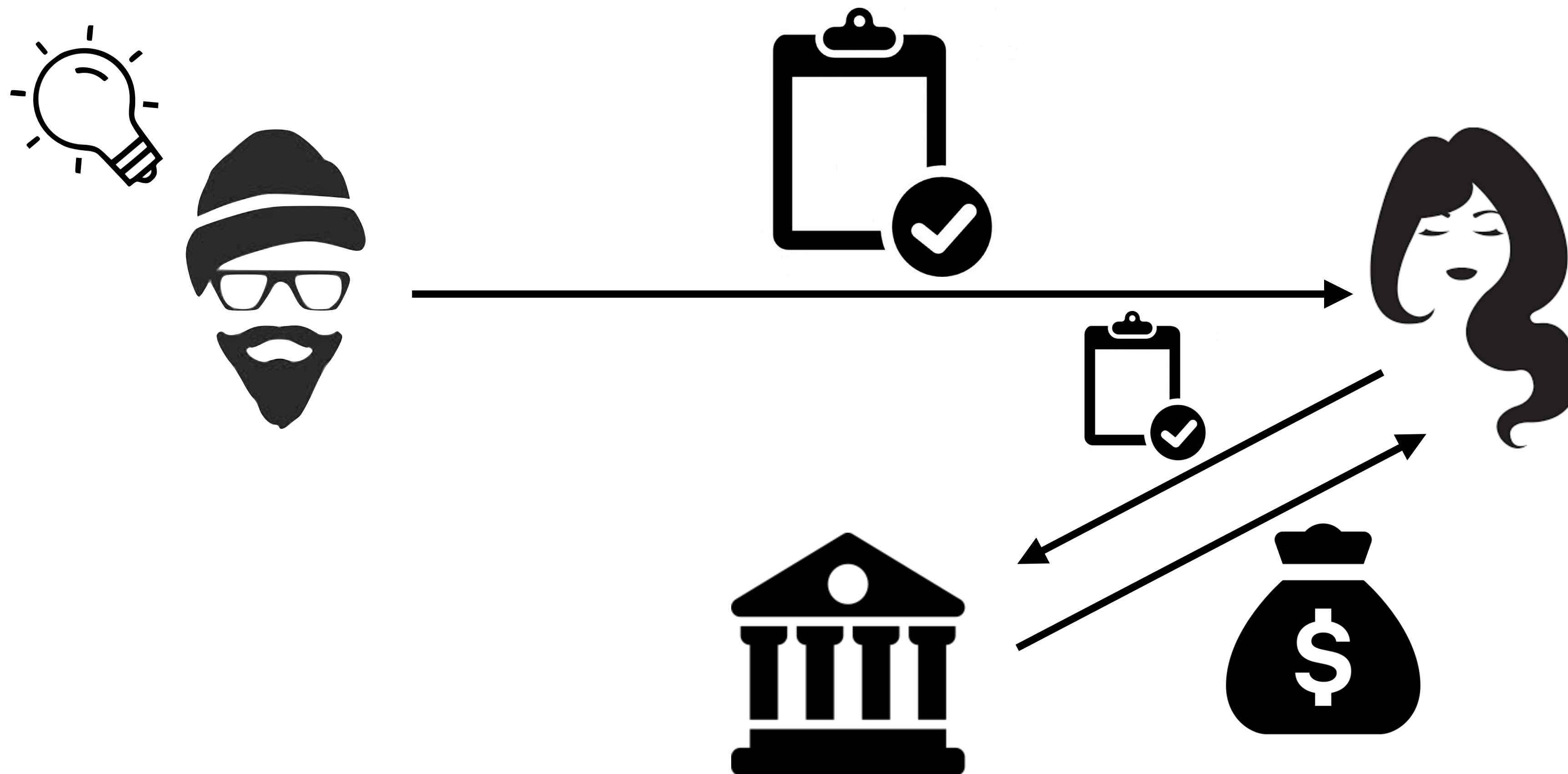


# Zero-Knowledge Proofs



# Second Tool: Zero-Knowledge Proofs

Scenario: you just solved a millenium problem, can you prove it to your friends?



# A Bit of Formalism: P, NP, Interactive Proofs

Some basics of complexity theory:

- A language  $L$  is a set of bitstrings.
- $P$  is the class of all languages which can be *decided* in polynomial time. More formally: a language  $\mathcal{L}$  is in  $P$  if there exists a polynomial-time Turing machine which, for any  $n$ , on input a bitstring  $x \in \{0,1\}^n$ , runs in time  $\text{poly}(n)$  and outputs 1 if and only if  $x$  belongs to  $\mathcal{L}$ .
- $NP$  is the class of all languages  $\mathcal{L}$  which admit an *efficient* (i.e. polytime) *proof of membership*. More formally: a language  $\mathcal{L}$  is in  $NP$  if there exists a polynomial-time Turing machine  $M$  such that

$$\mathcal{L} = \{x \in \{0,1\}^* : \exists w, |w| = \text{poly}(|x|) \wedge M(x, w) = 1\}.$$

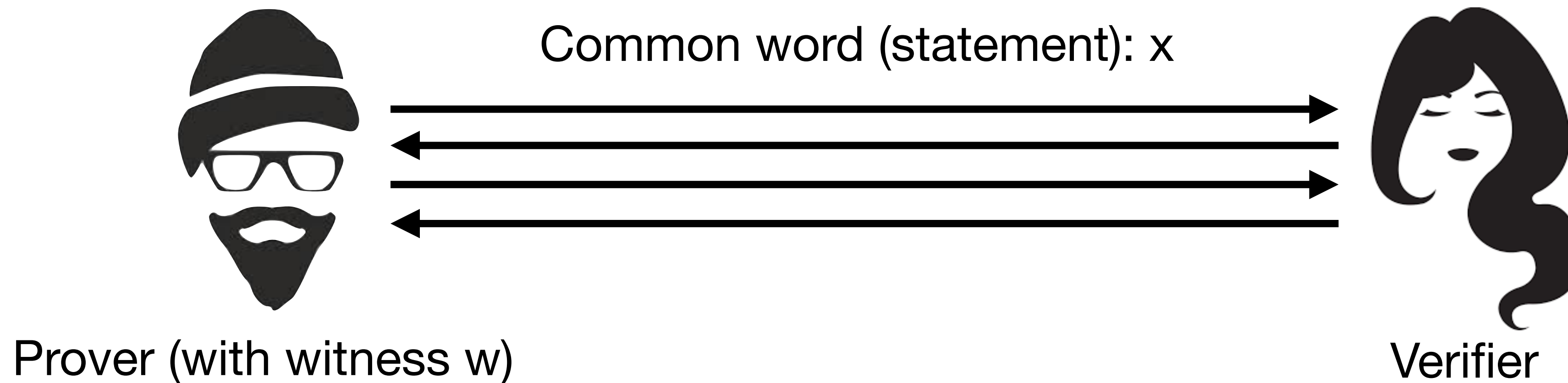
Intuitively, the statements of the form ‘ $x$  belongs to  $\mathcal{L}$ ’ for some  $NP$  language  $\mathcal{L}$  capture all *efficiently verifiable* statements.

- A language  $\mathcal{L}$  is *NP-complete* if  $\mathcal{L}$  is in  $NP$ , and the existence of a polytime algorithm for  $\mathcal{L}$  implies the existence of a poly time algorithm for *any language of NP*.

# Zero-Knowledge Proofs: Definition

A zero-knowledge proof of knowledge (ZKPoK) provides a mechanism to demonstrate that you know the proof of a statement (e.g. a theorem), without revealing *anything* beyond the fact that the statement is true (and that you know a proof).

More formally: a zero-knowledge proof is an *interactive protocol* between a *prover* and a *verifier*.



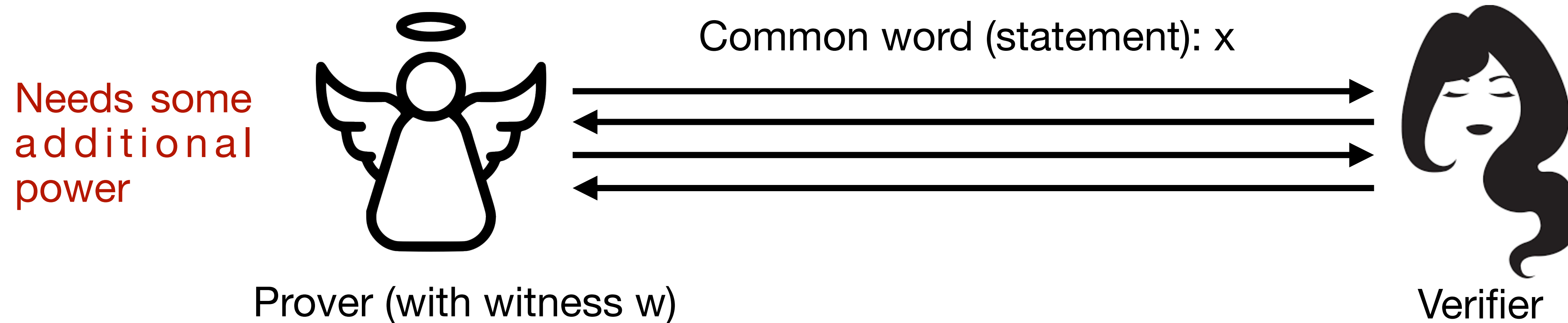
A zero-knowledge proof for 'x belongs to the language  $\mathcal{L}$ ' must satisfy three properties:

- **Correctness:** if indeed x belong to  $\mathcal{L}$  and the prover (with w) follows the protocol, the verifier accepts.
- **Soundness:** if x does not belong to  $\mathcal{L}$ , no cheating prover can cause the verifier to accept.
- **Zero-knowledge:** there is a simulator which, for any x in  $\mathcal{L}$ , can simulate (without w) the interaction

# Zero-Knowledge Proofs: Definition

A zero-knowledge proof of knowledge (ZKPoK) provides a mechanism to demonstrate that you know the proof of a statement (e.g. a theorem), without revealing *anything* beyond the fact that the statement is true (and that you know a proof).

More formally: a zero-knowledge proof is an *interactive protocol* between a *prover* and a *verifier*.



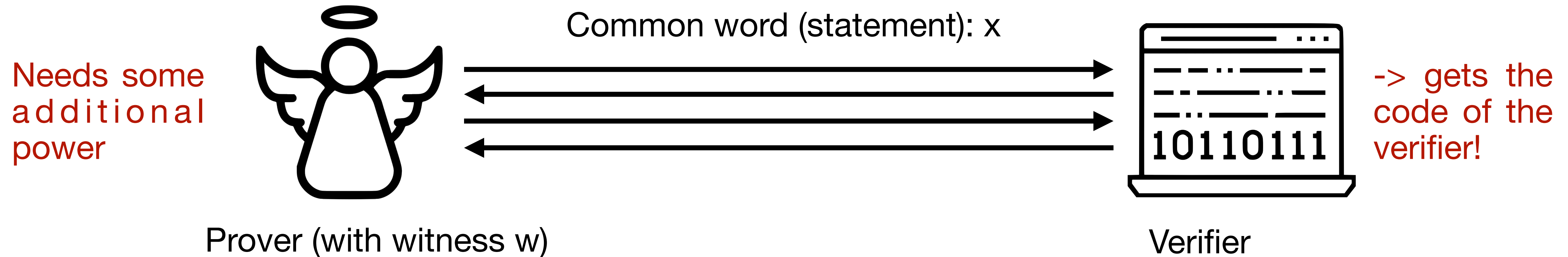
A zero-knowledge proof for 'x belongs to the language  $\mathcal{L}$ ' must satisfy three properties:

- **Correctness:** if indeed x belong to  $\mathcal{L}$  and the prover (with w) follows the protocol, the verifier accepts.
- **Soundness:** if x does not belong to  $\mathcal{L}$ , no cheating prover can cause the verifier to accept.
- **Zero-knowledge:** there is a simulator which, for any x in  $\mathcal{L}$ , can simulate (without w) the interaction

# Zero-Knowledge Proofs: Definition

A zero-knowledge proof of knowledge (ZKPoK) provides a mechanism to demonstrate that you know the proof of a statement (e.g. a theorem), without revealing *anything* beyond the fact that the statement is true (and that you know a proof).

More formally: a zero-knowledge proof is an *interactive protocol* between a *prover* and a *verifier*.



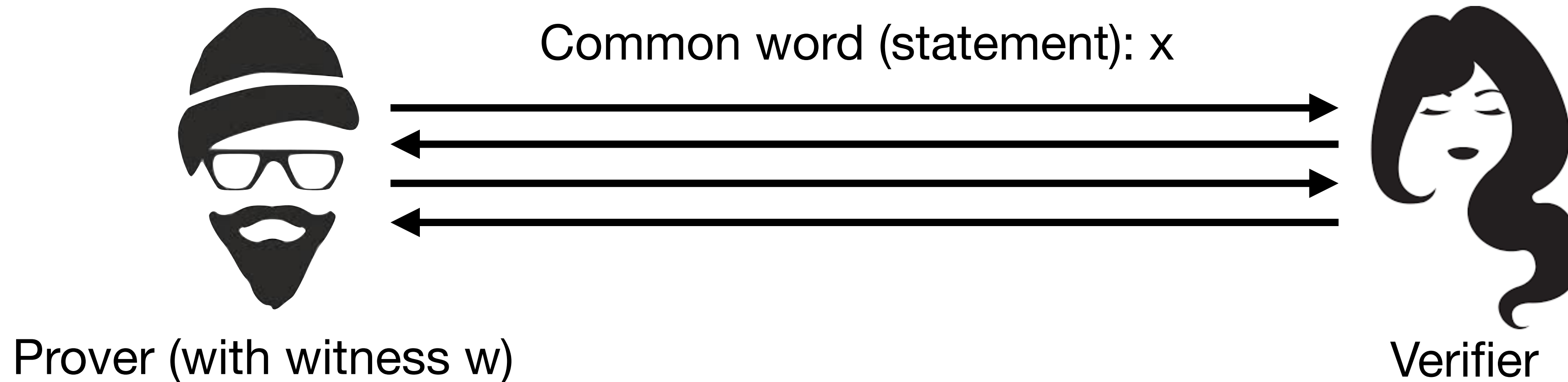
A zero-knowledge proof for 'x belongs to the language  $\mathcal{L}$ ' must satisfy three properties:

- **Correctness:** if indeed x belong to  $\mathcal{L}$  and the prover (with w) follows the protocol, the verifier accepts.
- **Soundness:** if x does not belong to  $\mathcal{L}$ , no cheating prover can cause the verifier to accept.
- **Zero-knowledge:** there is a simulator which, for any x in  $\mathcal{L}$ , can simulate (without w) the interaction

# Zero-Knowledge Proofs of Knowledge: Definition

A zero-knowledge proof of knowledge (ZKPoK) solves our problem, and much more: it provides a mechanism to demonstrate that you know the proof of a statement (e.g. a theorem), without revealing *anything* beyond the fact that the statement is true (and that you know a proof).

More formally: a zero-knowledge proof is an *interactive protocol* between a *prover* and a *verifier*.

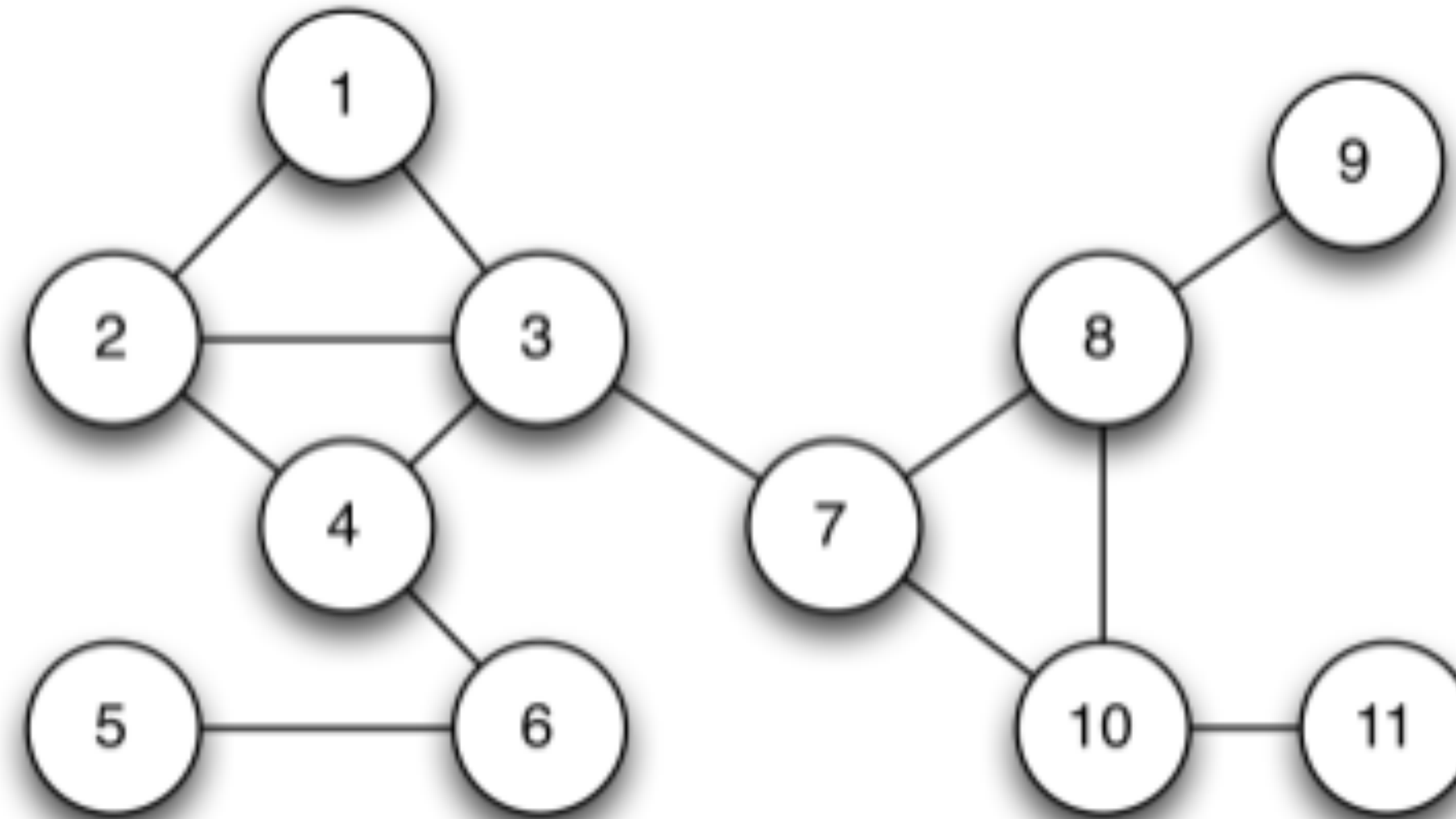


A zero-knowledge proof for 'x belongs to the language  $\mathcal{L}$ ' must satisfy three properties:

- **Correctness:** if indeed x belong to  $\mathcal{L}$  and the prover (with w) follows the protocol, the verifier accepts.
- **Extractability:** There exists an extractor which, by *rewinding* the prover, can extract the witness
- **Zero-knowledge:** there is a simulator which, for any x in  $\mathcal{L}$ , can simulate (without w) the interaction

# Example: Deploying a Cellular Network

You are a telecom company deploying a new cellular communications network. The graph below is the network structure: nodes are radio towers, edges indicate overlaps. To avoid transmission interferences, each tower can be configured to one of three different frequencies.



**Challenge:** assigning a frequency to each tower such that no interference can occur. Some of you might have recognized an instance of the *3-coloring* graph problem.

---

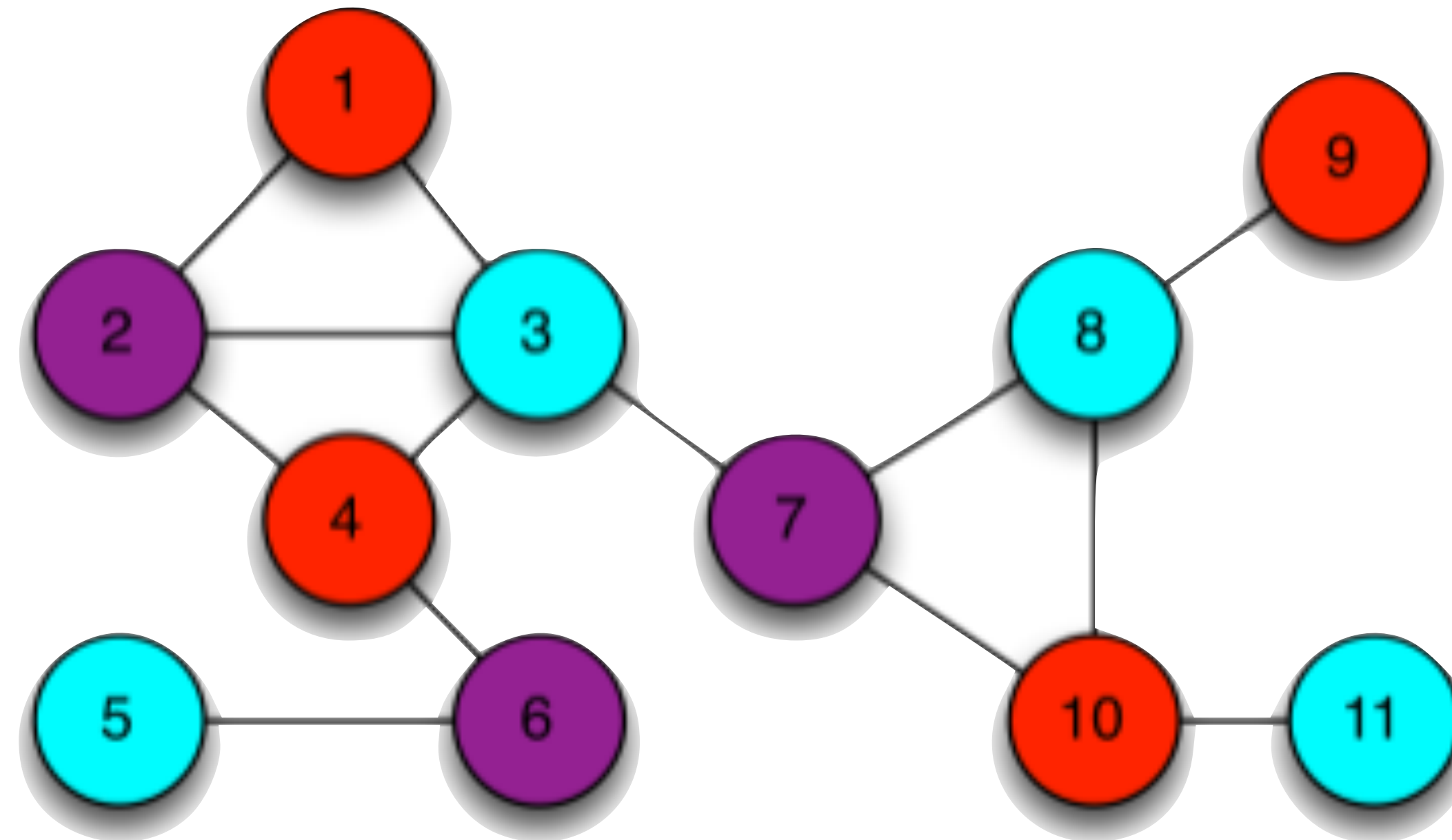
Example taken from a great blog post by Matthew Green:

<https://blog.cryptographyengineering.com/2014/11/27/zero-knowledge-proofs-illustrated-primer/>



# Example: Deploying a Cellular Network

You are a telecom company deploying a new cellular communications network. The graph below is the network structure: nodes are radio towers, edges indicate overlaps. To avoid transmission interferences, each tower can be configured to one of three different frequencies.



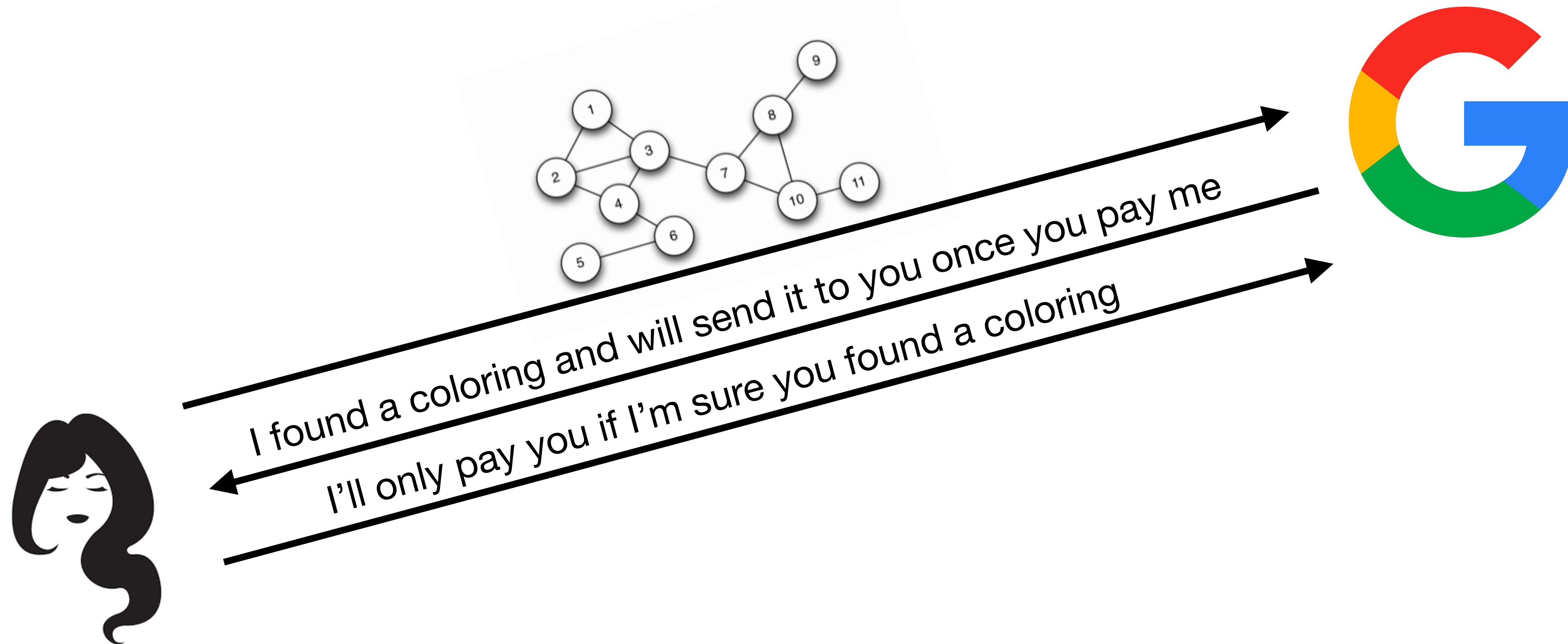
**Challenge:** assigning a frequency to each tower such that no interference can occur. Some of you might have recognized an instance of the *3-coloring* graph problem.

---

Example taken from a great blog post by Matthew Green:

<https://blog.cryptographyengineering.com/2014/11/27/zero-knowledge-proofs-illustrated-primer/>

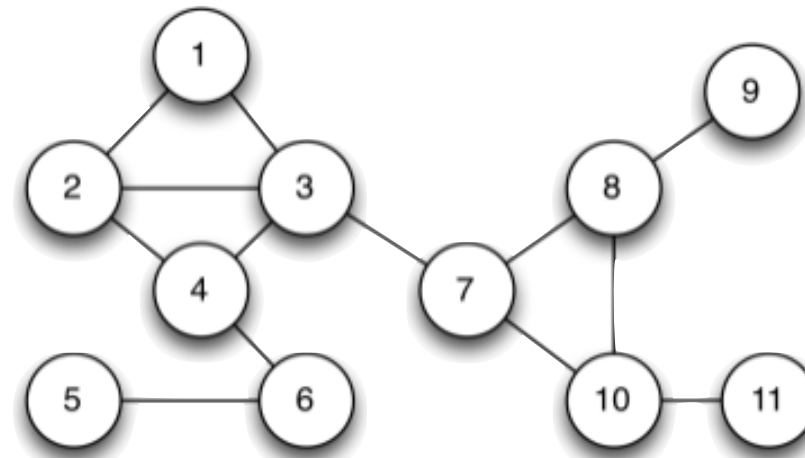
# Example: Deploying a Cellular Network



Example taken from a great blog post by Matthew Green:

<https://blog.cryptographyengineering.com/2014/11/27/zero-knowledge-proofs-illustrated-primer/>

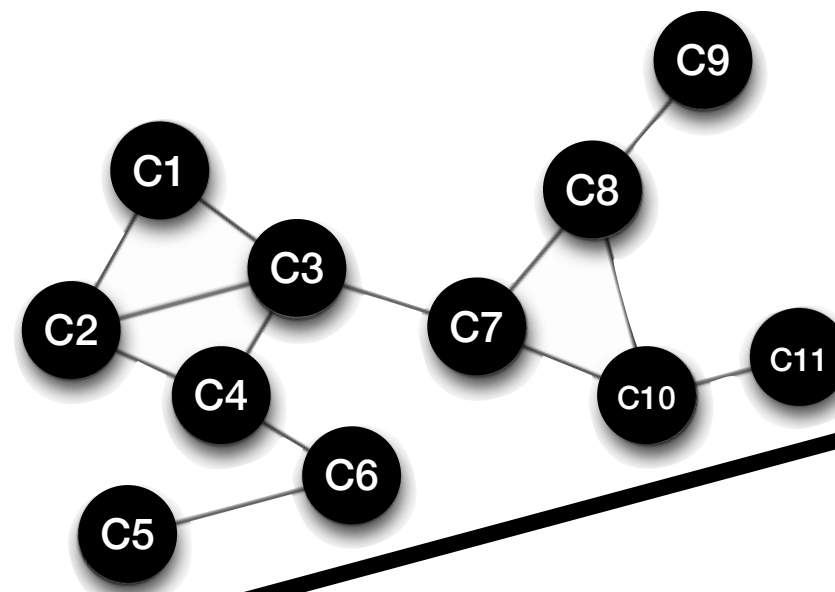
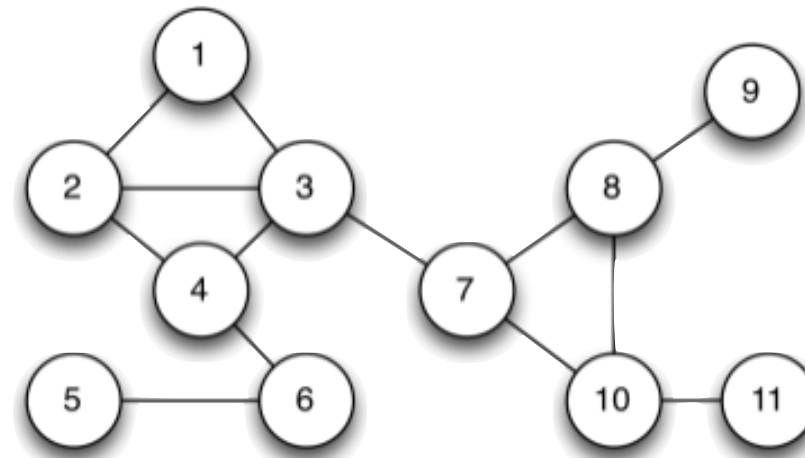
# A Zero-Knowledge Protocol



Example taken from a great blog post by Matthew Green:

<https://blog.cryptographyengineering.com/2014/11/27/zero-knowledge-proofs-illustrated-primer/>

# A Zero-Knowledge Protocol



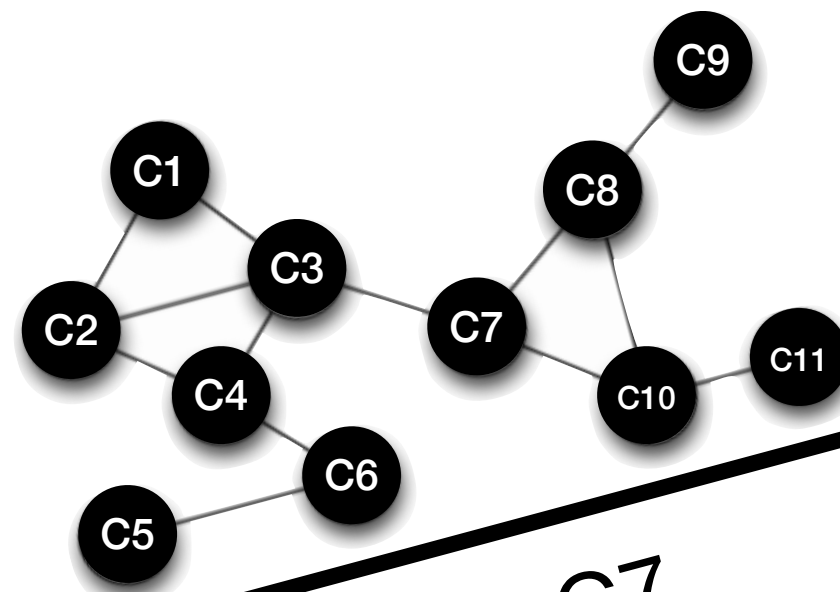
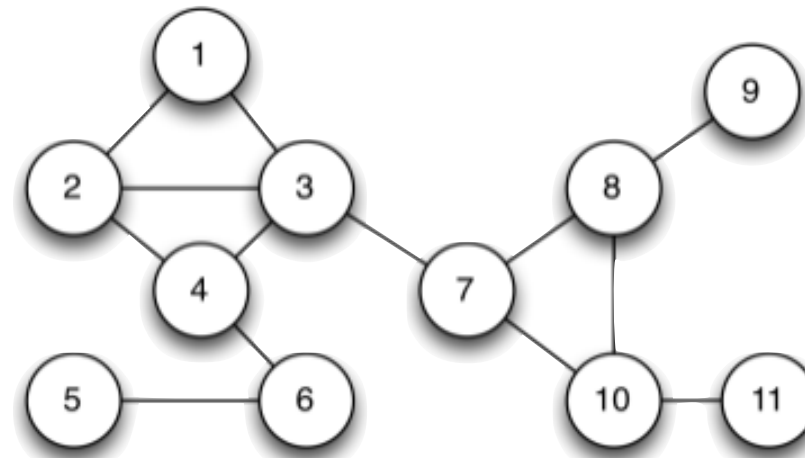
- Pick an association: colors {red, blue, purple}  $\leftrightarrow$  letters {A, B, C}
- Find a coloring and commit to the letters for each node.



Example taken from a great blog post by Matthew Green:

<https://blog.cryptographyengineering.com/2014/11/27/zero-knowledge-proofs-illustrated-primer/>

# A Zero-Knowledge Protocol



Open C3 - C7



- Pick an edge and challenge Google to open the nodes.

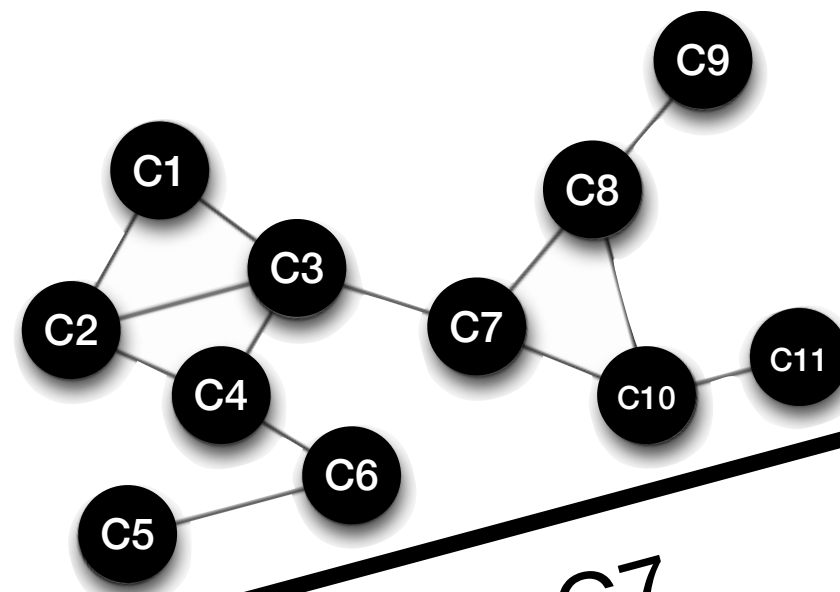
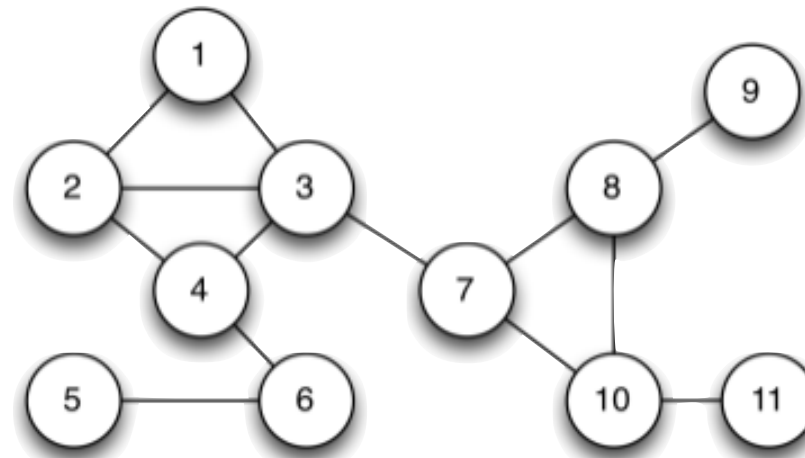


- Pick an association: colors {red, blue, purple}  $\leftrightarrow$  letters {A, B, C}
- Find a coloring and commit to the letters for each node.

Example taken from a great blog post by Matthew Green:

<https://blog.cryptographyengineering.com/2014/11/27/zero-knowledge-proofs-illustrated-primer/>

# A Zero-Knowledge Protocol



Open C3 - C7

('A', d3), ('B', d7)

- Pick an edge and challenge Google to open the nodes.

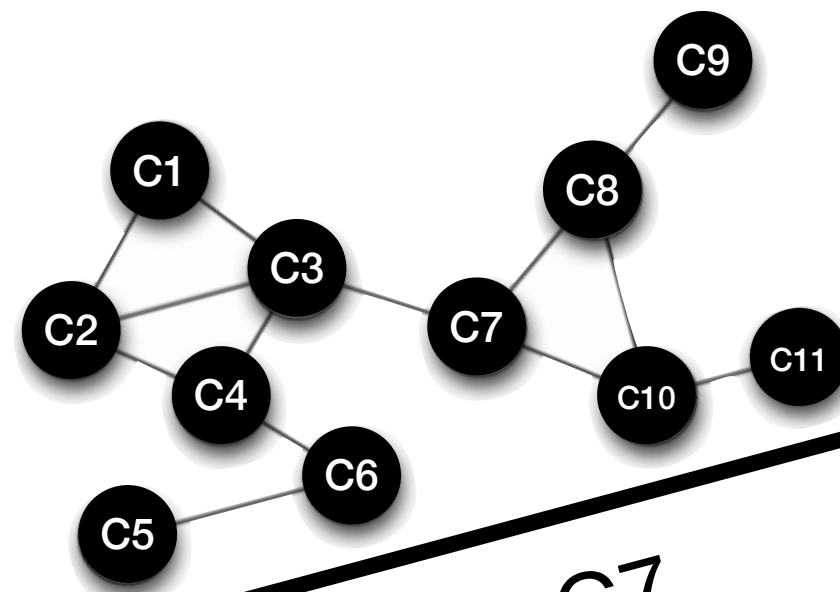
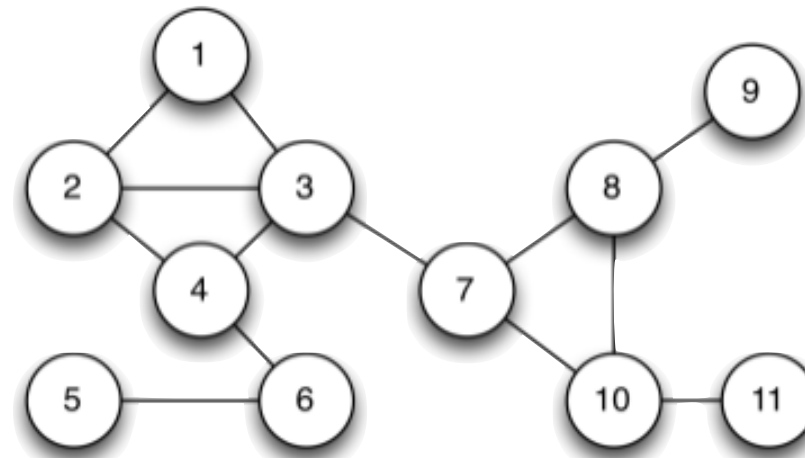


- Pick an association: colors {red, blue, purple}  $\leftrightarrow$  letters {A, B, C}
- Find a coloring and commit to the letters for each node.
- Send the opening to Alice's query.

Example taken from a great blog post by Matthew Green:

<https://blog.cryptographyengineering.com/2014/11/27/zero-knowledge-proofs-illustrated-primer/>

# A Zero-Knowledge Protocol



Open C3 - C7

('A', d3), ('B', d7)

- Pick an edge and challenge Google to open the nodes.



- Pick an association: colors {red, blue, purple}  $\leftrightarrow$  letters {A, B, C}
- Find a coloring and commit to the letters for each node.
- Send the opening to Alice's query.

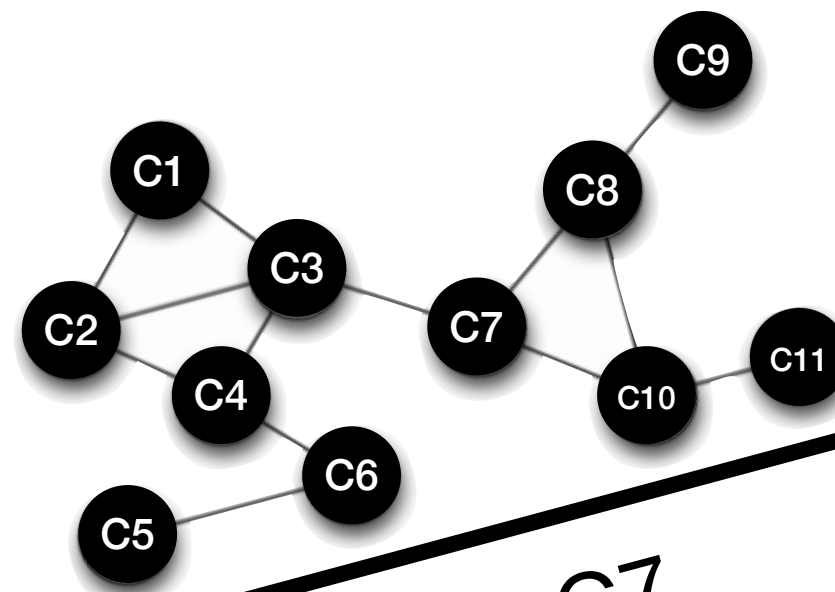
- Accept if  $\text{Verify}(\text{'letter3'}, d3) = \text{Verify}(\text{'letter7'}, d7) = 1$  and  $\text{letter3} \neq \text{letter7}$ ; otherwise, reject.

Example taken from a great blog post by Matthew Green:

<https://blog.cryptographyengineering.com/2014/11/27/zero-knowledge-proofs-illustrated-primer/>

# A Zero-Knowledge Protocol

Correctness: check it.



Open C3 - C7

('A', d3), ('B', d7)

- Pick an edge and challenge Google to open the nodes.



- Pick an association: colors {red, blue, purple}  $\leftrightarrow$  letters {A, B, C}
- Find a coloring and commit to the letters for each node.
- Send the opening to Alice's query.

- Accept if  $\text{Verify}(\text{'letter3'}, d3) = \text{Verify}(\text{'letter7'}, d7) = 1$  and  $\text{letter3} \neq \text{letter7}$ ; otherwise, reject.

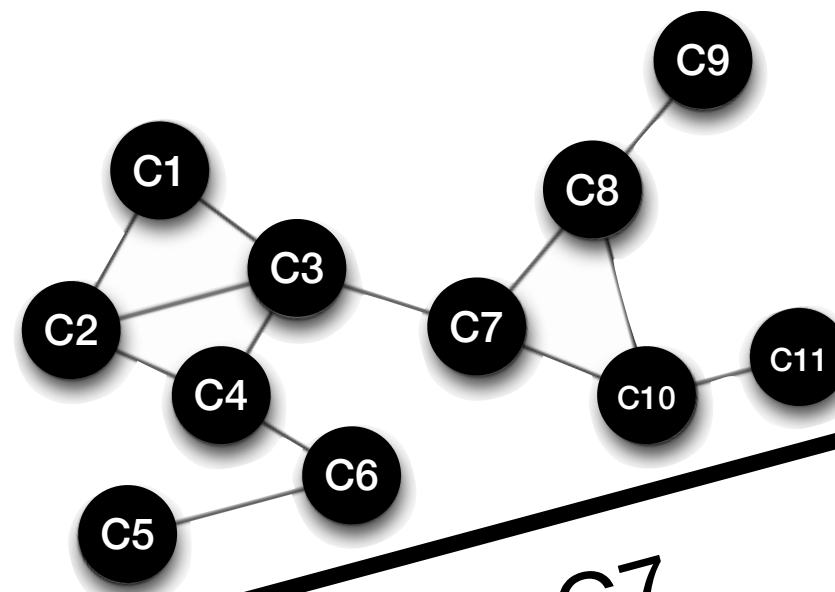
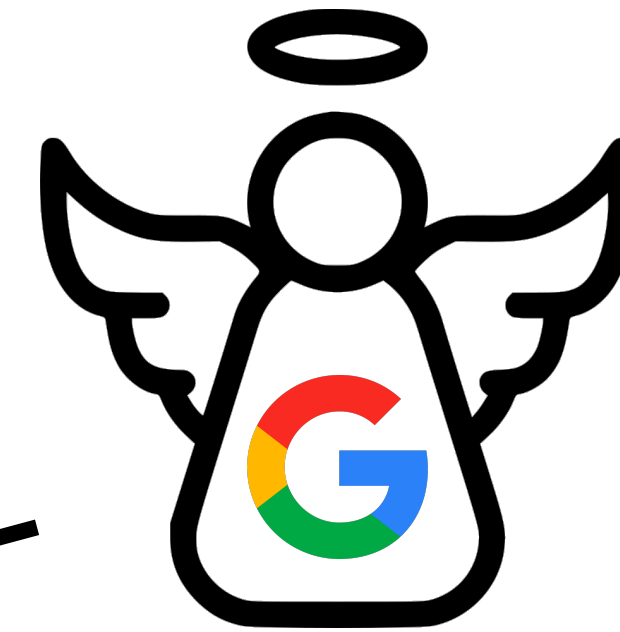
Example taken from a great blog post by Matthew Green:

<https://blog.cryptographyengineering.com/2014/11/27/zero-knowledge-proofs-illustrated-primer/>



# A Zero-Knowledge Protocol

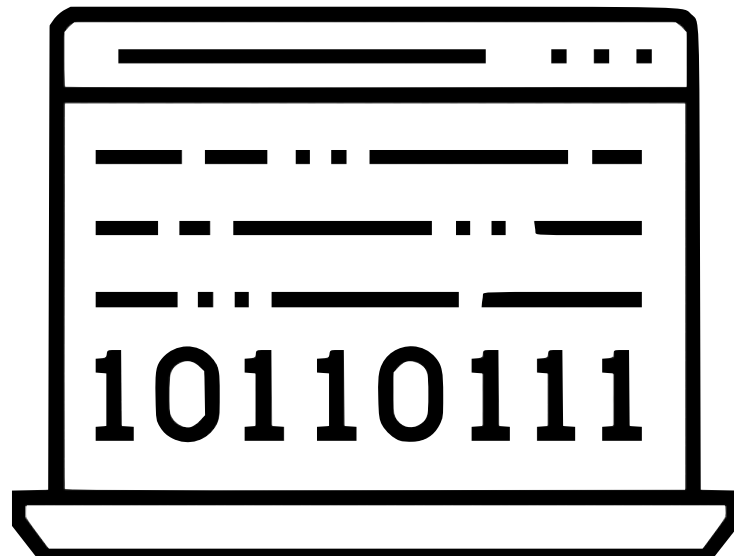
Zero-Knowledge



Open C3 - C7

('A', d3), ('B', d7)

- Pick an edge and challenge Google to open the nodes.



- Pick a random edge, commit to two random letter in {A,B,C} for this edge, and 0 for all other nodes.

- If Alice's code return this edge, open it, otherwise, restart.

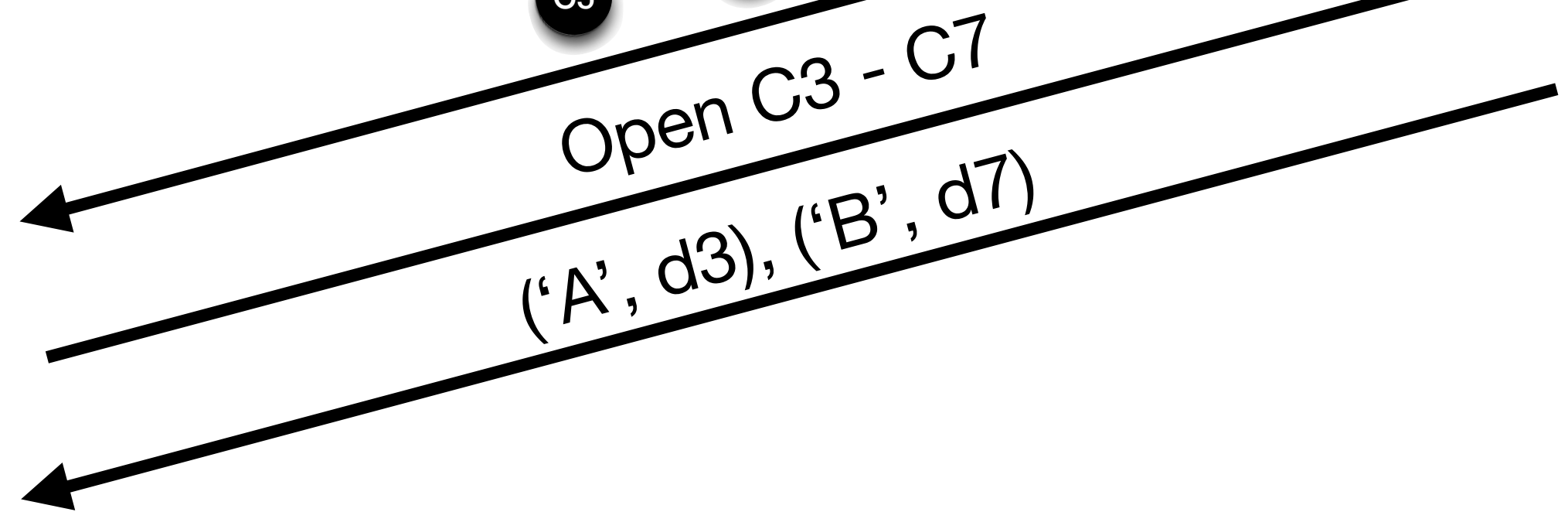
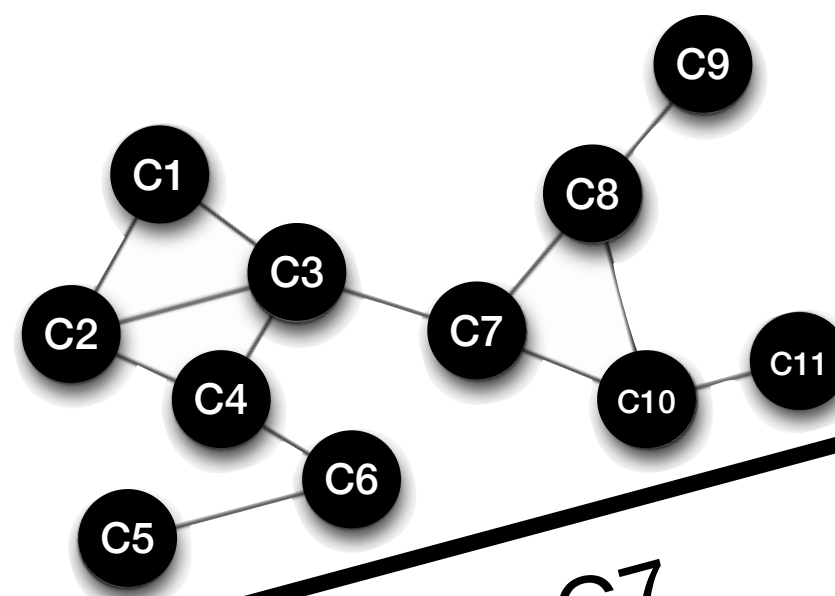
Convince yourself that this produces a transcript indistinguishable from a honest transcript

Example taken from a great blog post by Matthew Green:

<https://blog.cryptographyengineering.com/2014/11/27/zero-knowledge-proofs-illustrated-primer/>

# A Zero-Knowledge Protocol

Soundness:  $1/|E|$



- Since Google does *not* know a 3-coloring, there must exist an edge between nodes associated to the same letter
- Alice picks this edge with prob  $1/|E|$
- Google cannot open this edge to different letters without breaking the binding property of the commitment.

Example taken from a great blog post by Matthew Green:

<https://blog.cryptographyengineering.com/2014/11/27/zero-knowledge-proofs-illustrated-primer/>

# A Zero-Knowledge Protocol

Repeat all  $n \cdot |E|$  times

$$\left(1 - \frac{1}{|E|}\right)^{n \cdot |E|} = e^{-n}$$



- Since Google does *not* know a 3-coloring, there must exist an edge between nodes associated to the same letter
- Alice picks this edge with prob  $1/|E|$
- Google cannot open this edge to different letters without breaking the binding property of the commitment.

Example taken from a great blog post by Matthew Green:

<https://blog.cryptographyengineering.com/2014/11/27/zero-knowledge-proofs-illustrated-primer/>

# A Zero-Knowledge Protocol for *any* Statement

- We just gave a zero-knowledge proof system for membership to the 3-colorable language
- 3-coloring is NP-complete: *any* problem in NP can be transformed (in polynomial time) into an instance of the 3-coloring problem
- This gives a zero-knowledge proof for any problem in NP: take the target NP problem, transform it into an instance of 3-coloring (the word is mapped to a graph and the NP-witness is mapped to a 3-coloring of the graph), and use the zero-knowledge proof to show knowledge of the witness.

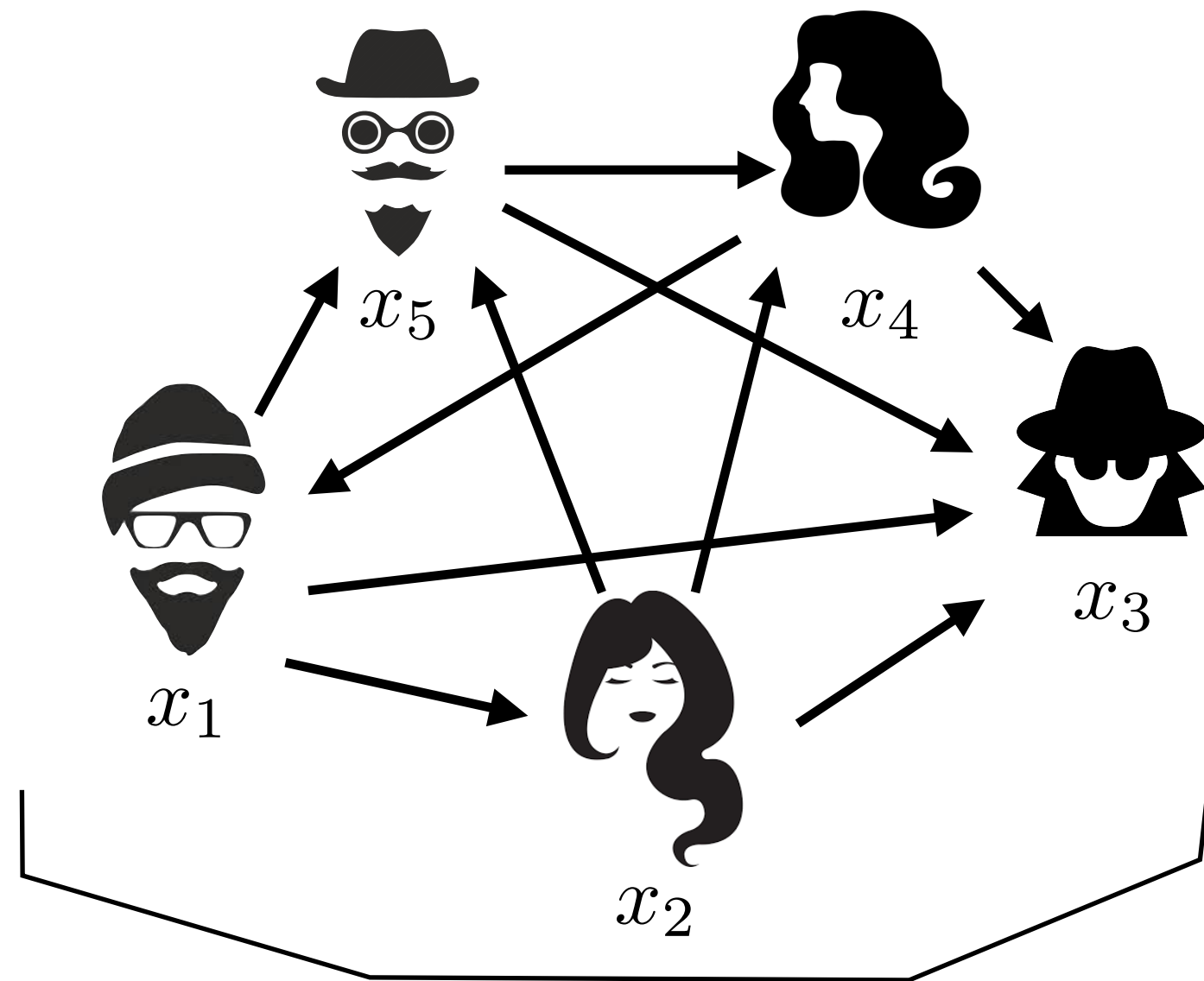
---

## Getting a zero-knowledge proof *of knowledge*

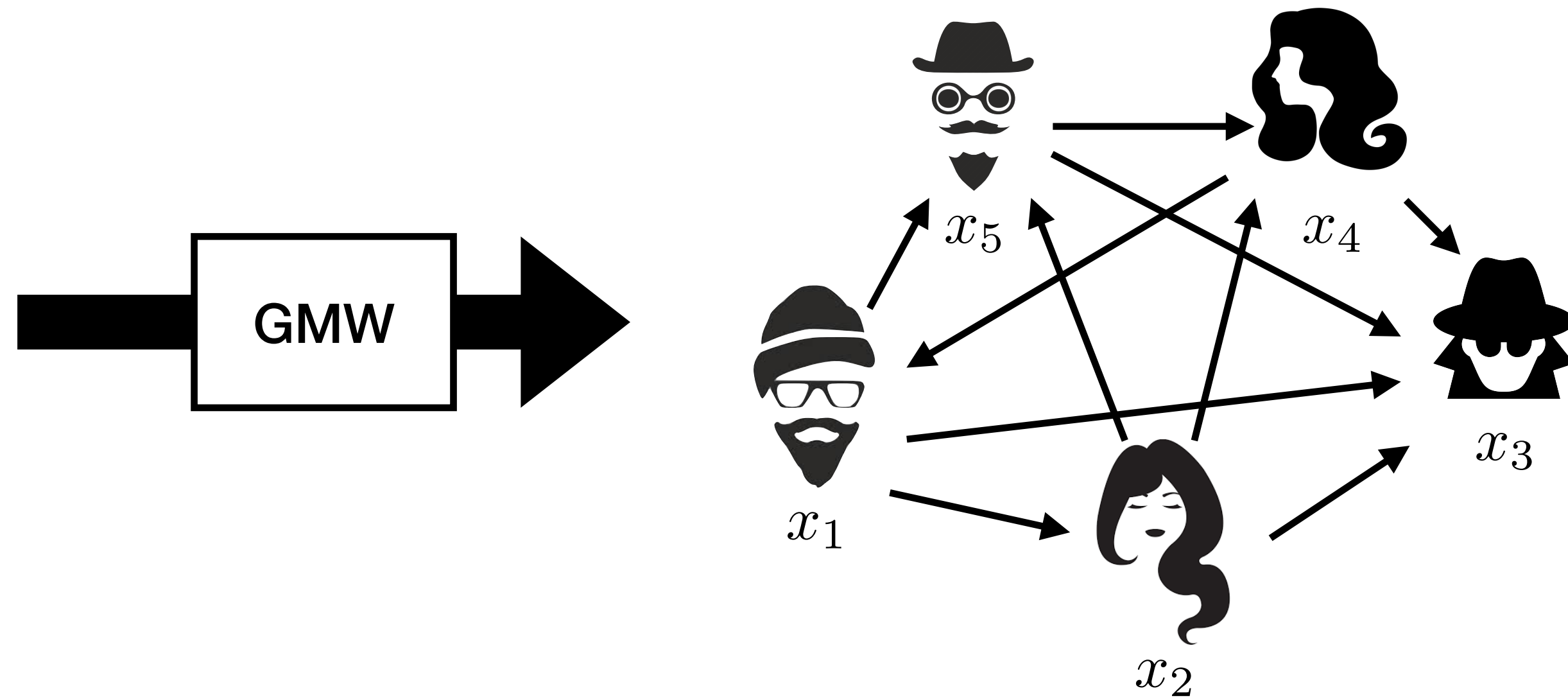
- We only gave a zero-knowledge proof of *membership*. To get a zero-knowledge proof of knowledge, we need a property stronger than soundness: *extractability*.
- There are standard techniques to achieve this stronger property. Example: use an *extractable* commitment (for example, a public key encryption scheme: the secret key is the *extraction trapdoor*).

# Wrapping Up: Secure Computation against Malicious Adversaries

## Honest-But Curious Protocol



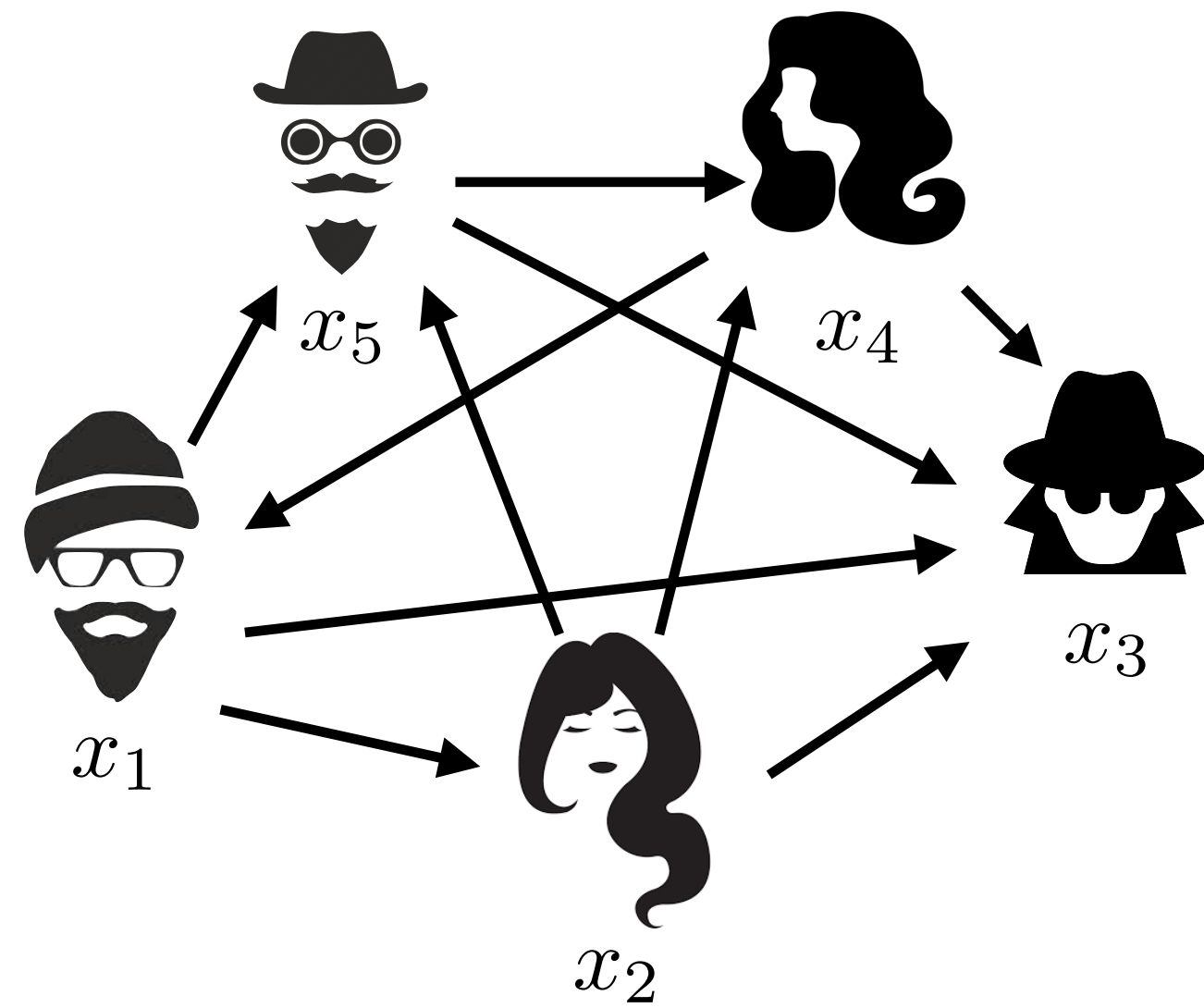
## Malicious Protocol



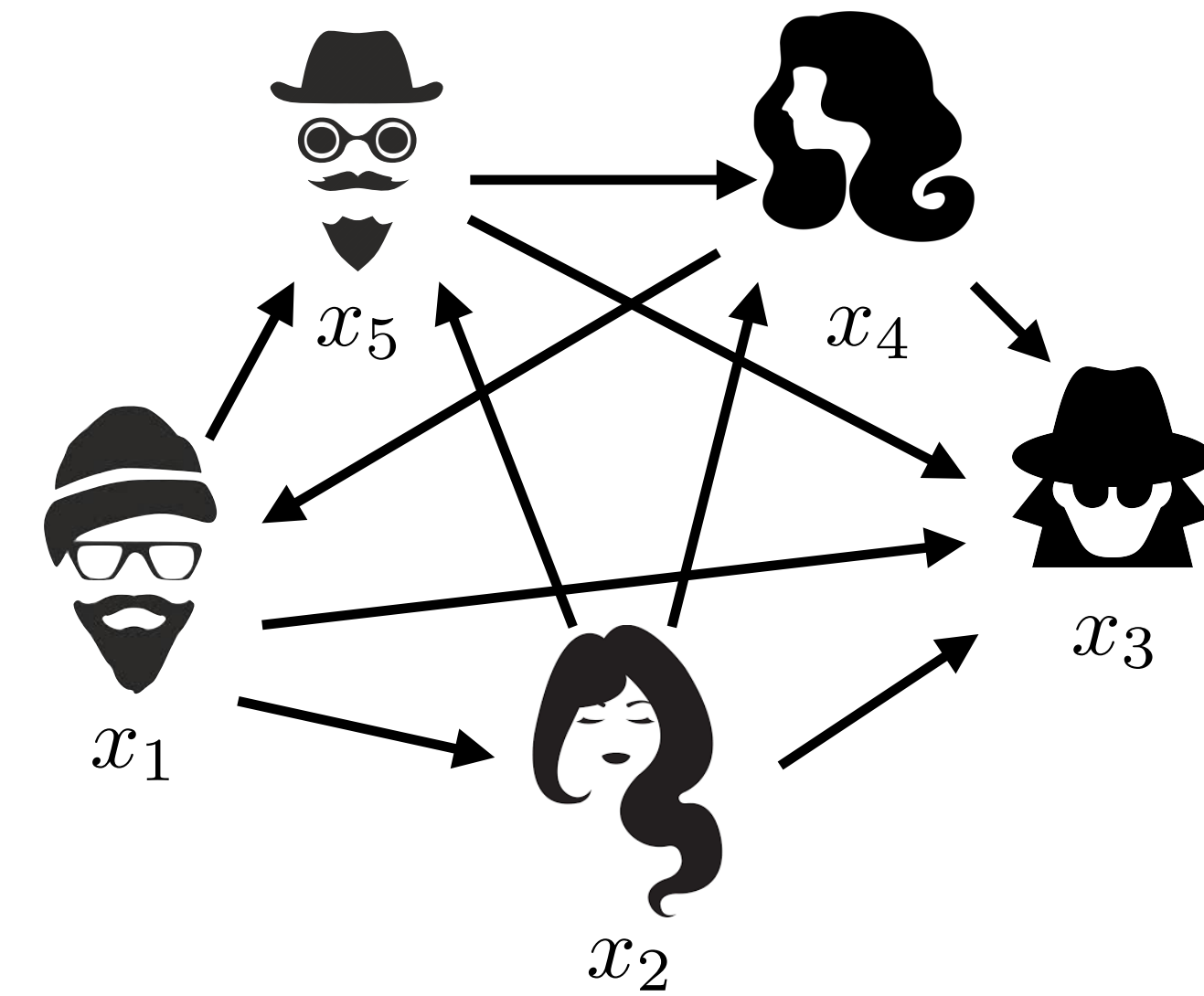
- The honest-but-curious protocol is in the **broadcast model**: each message is sent to everyone.
- Formalize the protocol as being a *function* NextMessage: NextMessage takes as input
  - The input of the current player
  - The random tape of the current player
  - The current transcript (i.e., the list of all message received during the protocol so far)and outputs the next message sent by the current player to all players.

# Wrapping Up: Secure Computation against Malicious Adversaries

## Honest-But Curious Protocol



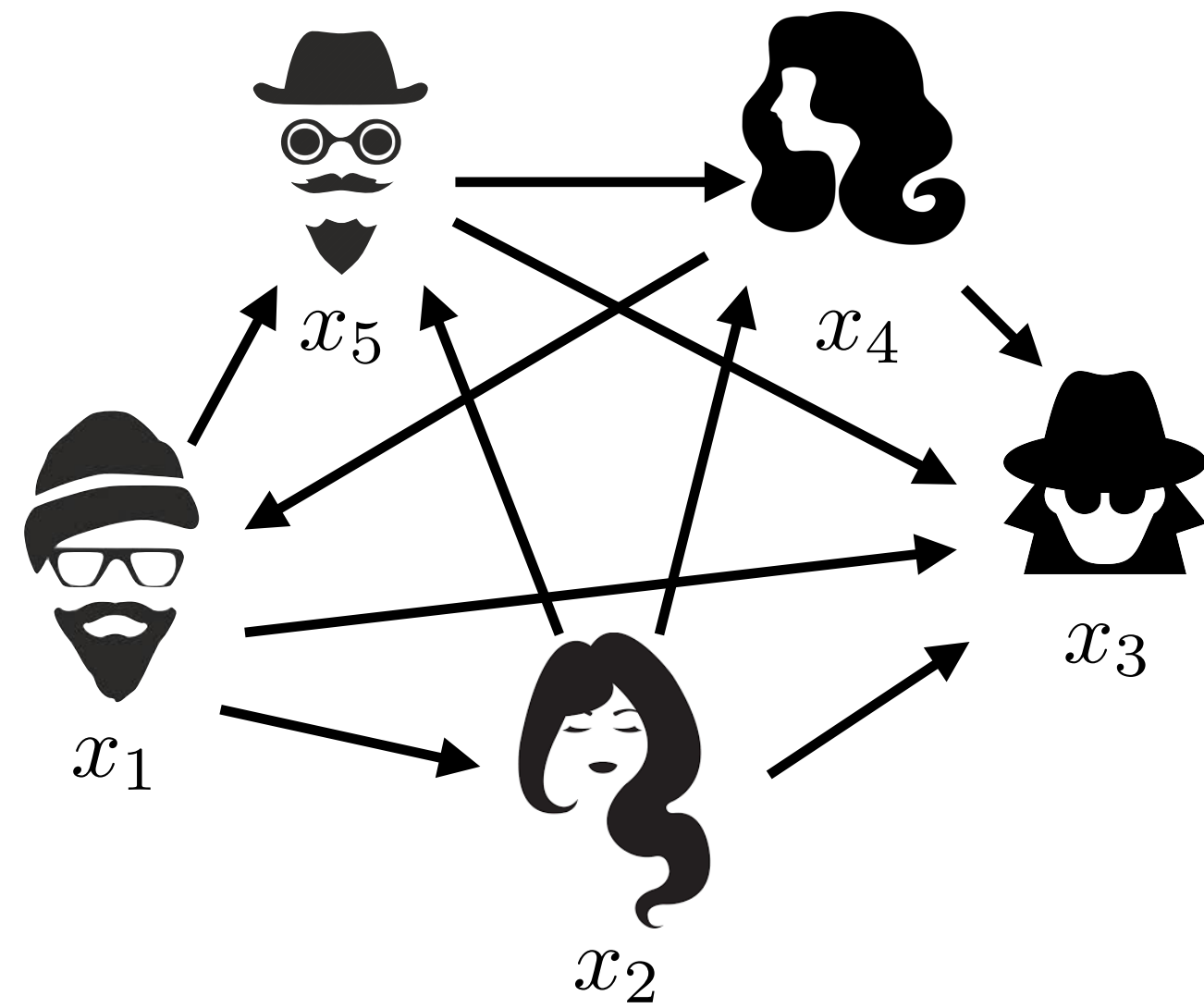
## Malicious Protocol



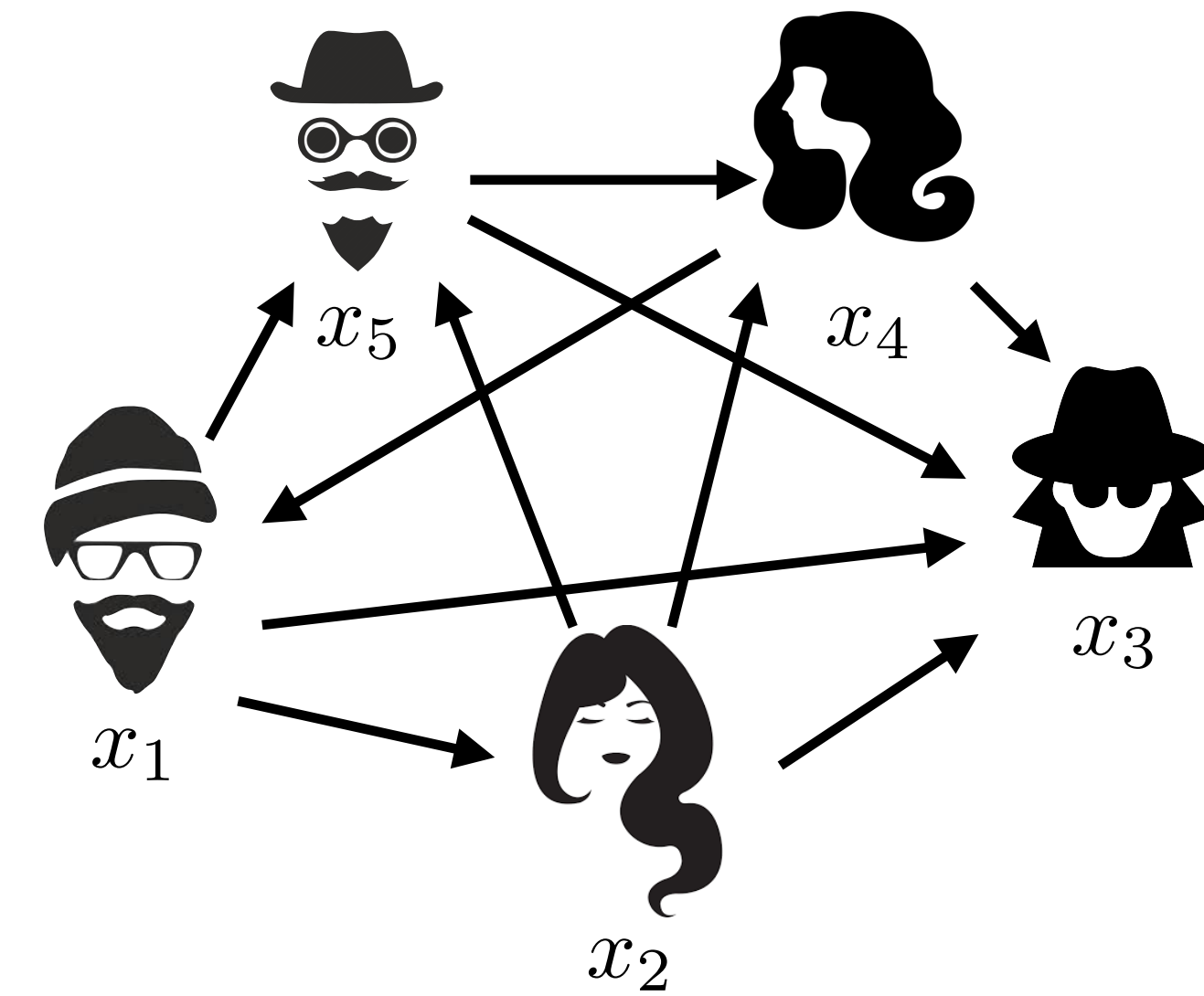
- Start from the honest-but-curious protocol. Let all parties use a **commitment** to commit to their input and their random tape before starting the protocol.
- Each time a player sends a message in the honest-but-curious protocol, he additionally performs a **zero-knowledge proof of knowledge** with *all other players in the protocol*, to prove the following statement: 'I know an input  $x$  and a random tape  $r$  such that (1) the commitments sent in the first flow are valid commitments to  $x$  and  $r$ , and (2) the message I just send is equal to  $\text{NextMessage}(x, r, \text{transcript})$ '

# Wrapping Up: Secure Computation against Malicious Adversaries

## Honest-But Curious Protocol



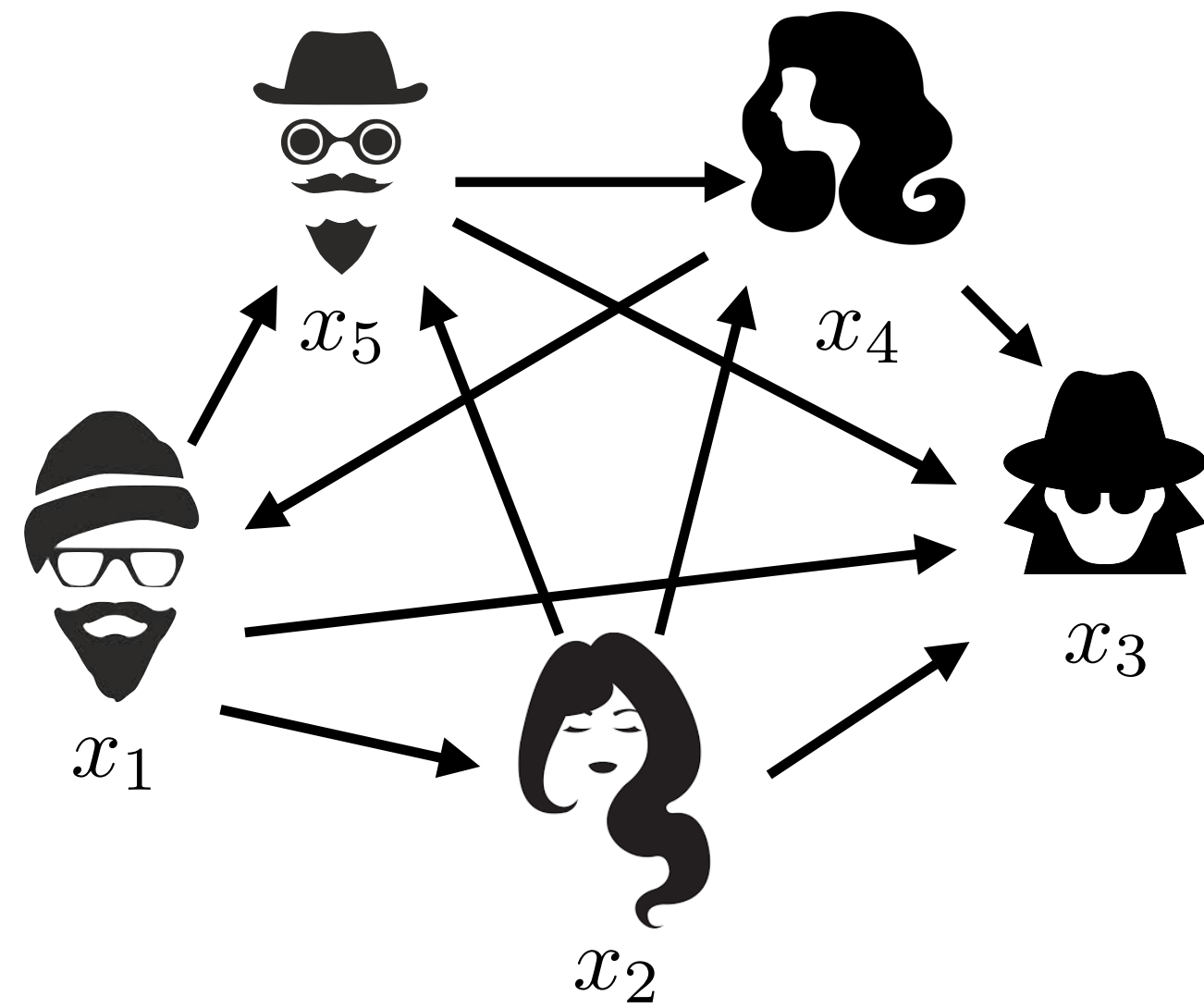
## Malicious Protocol



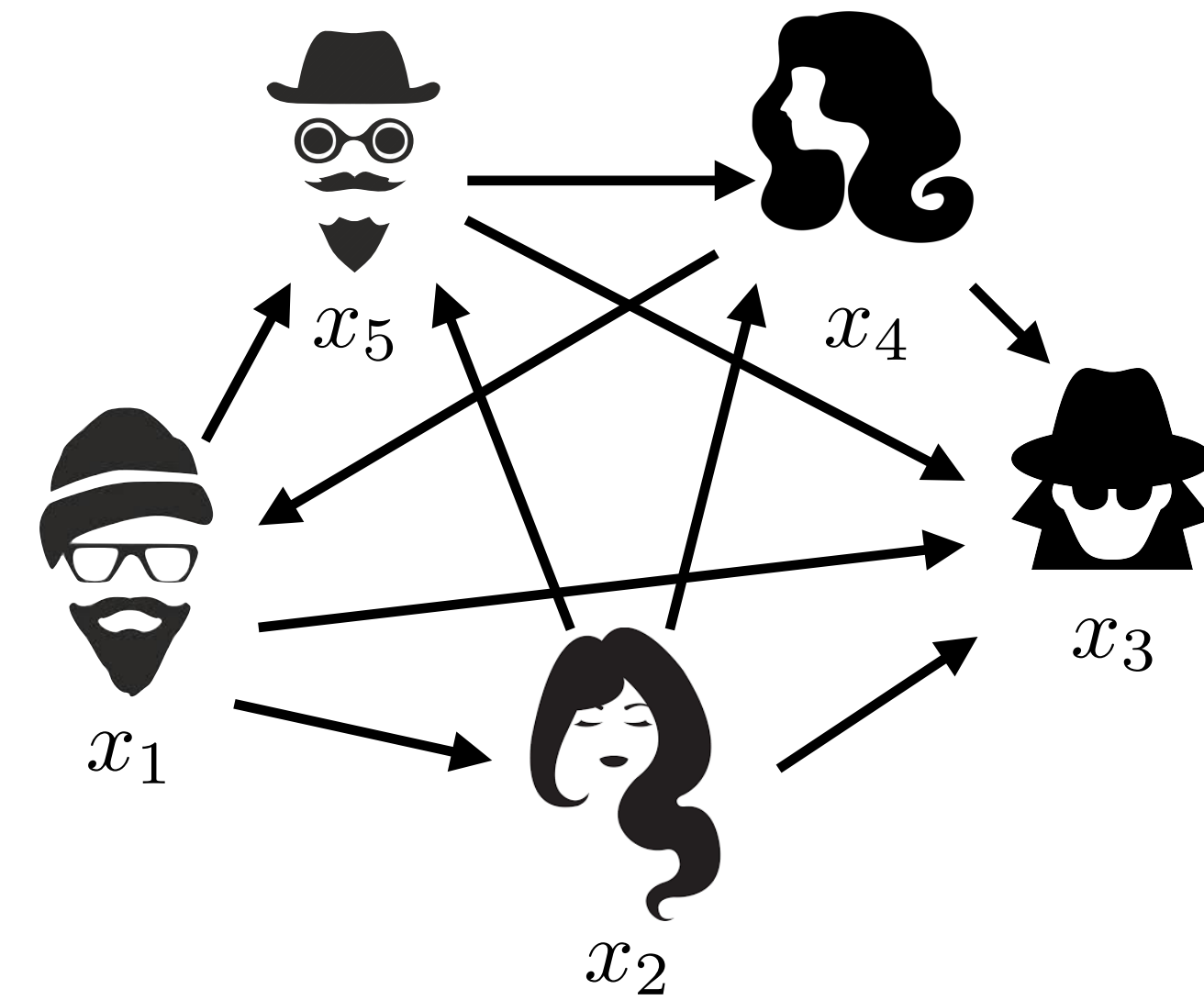
- Start from the honest-but-curious protocol. Let all parties use a **commitment** to commit to their input and their random tape before starting the protocol.
- Each time a player sends a message in the honest-but-curious protocol, he additionally performs a **zero-knowledge proof of knowledge** with *all other players in the protocol*, to prove the following statement: 'I know an input  $x$  and a random tape  $r$  such that (1) the commitments sent in the first flow are valid commitments to  $x$  and  $r$ , and (2) the message I just send is equal to  $\text{NextMessage}(x, r, \text{transcript})$  => **this is an NP statement with witness  $(x, r, \text{randomness\_commit})!$**

# Wrapping Up: Secure Computation against Malicious Adversaries

## Honest-But Curious Protocol



## Malicious Protocol

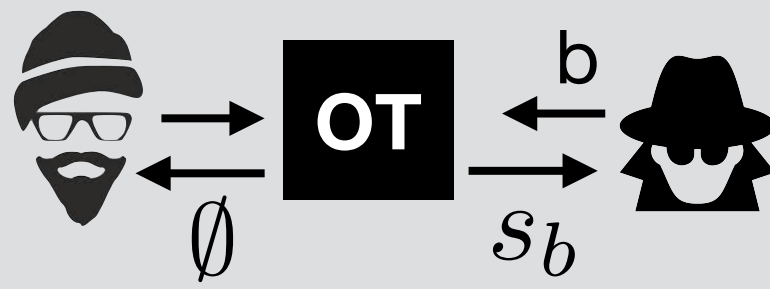


- Security (intuition): the zero-knowledge proofs do not harm the semi-honest security of the underlying protocol (proof: replace the parties by the ZK simulator). If the parties follow the protocol, we know that it securely emulates the functionality. But if any party ever deviates from the protocol, then by soundness of the ZK proof, he gets ‘caught’ by everyone, and the protocol terminates (it is equivalent to leaving the protocol, which is always an option).



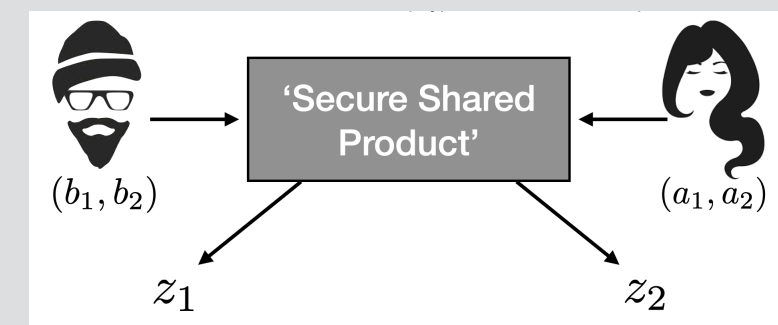
# Wrapping Up: Secure Computation against Malicious Adversaries

Use ElGamal to build public key encryption with oblivious keys

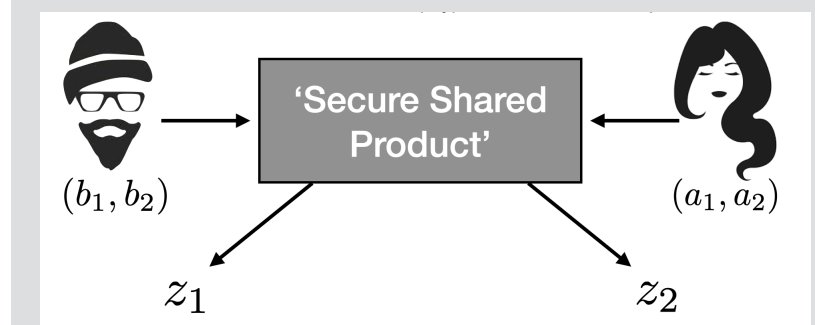


Build a semi-honest OT

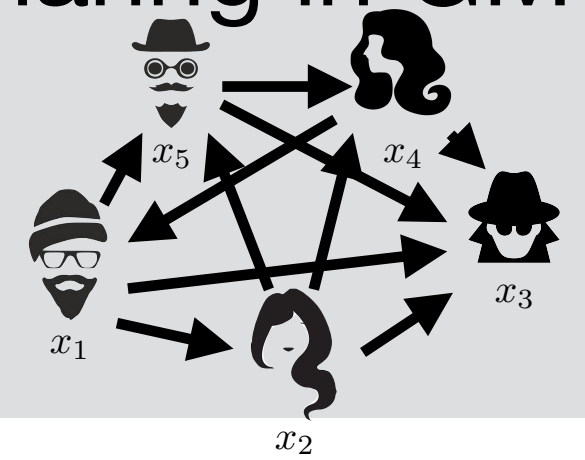
Build 2-party secure AND



Build n-party secure AND

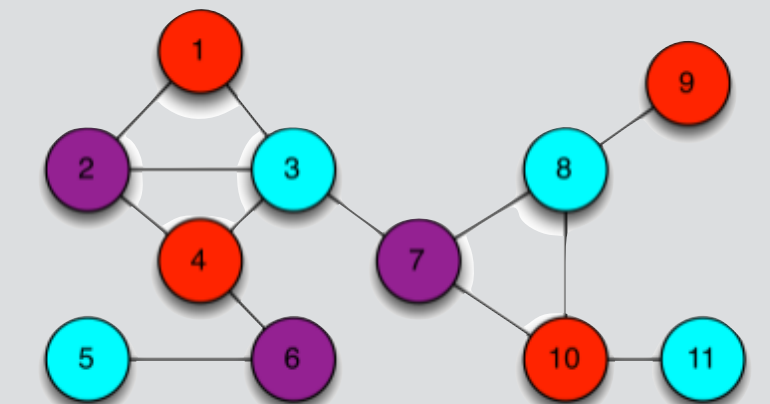


Use it + secret sharing in GMW



Use (e.g.) a PRG to build a commitment scheme

Build a ZK for NP



Use the GMW compiler on GMW



Bonus:  
same function to different functions

Bonus:  
deterministic to randomized

# That's all for today!

If you have any question after the lesson:

[couteau@irif.fr](mailto:couteau@irif.fr)